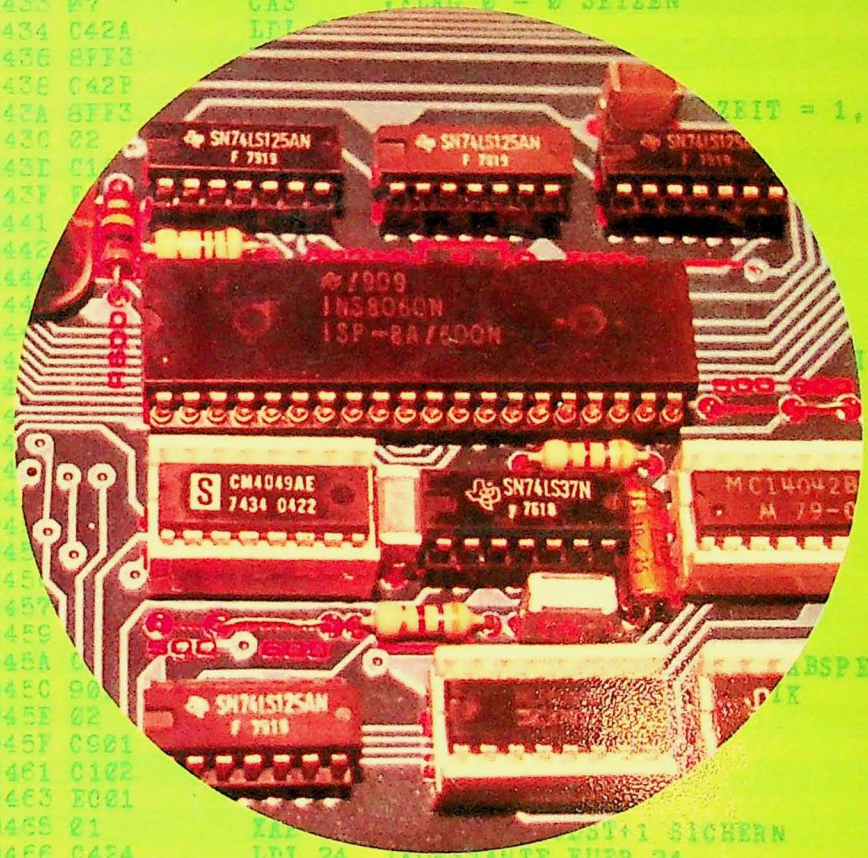
TOPP

M. Reinhard

mikroprozessor scmp II



von den Grundlagen
über die Bausteinfunktion
zum fertigen Programm

TOPP

Michael Reinhard

Mikroprozessor SCMP II

Von den Grundlagen
über die Bausteinfunktion
zum fertigen Programm

Frech Verlag Stuttgart

Der Verlag bittet zu beachten:
Unsere Veröffentlichungen von Schaltungen und Verfahren erfolgen ohne Rücksicht auf bestehende Patente, da sie nur für Amateur- und Lehrzwecke bestimmt sind. Eine gewerbliche Nutzung ist ausdrücklich untersagt.
Trotz sorgfältiger Überprüfung aller Schaltungen und Angaben durch Verfasser und Verlag lassen sich Fehler nie ganz vermeiden. Der Verlag kann deshalb weder eine Garantie für Fehlerlosigkeit noch die juristische Verantwortung oder irgendeine Haftung übernehmen. Verfasser und Verlag sind für Hinweise auf Fehler sowie für Verbesserungs- oder Ergänzungsvorschläge dankbar.

ISBN 3-7724-0459-6 Best.-Nr. 475

© 1982

frech-verlag
GmbH+Co. Druck KG Stuttgart

Druck: Frech Stuttgart
Satz: Fotosatz Hauer, Stuttgart

Wissenschaftliche Assistenz:
Professor Otmar Kilgenstein

Inhaltsverzeichnis

Vorwort 6

1. Fachwortlexikon 7
1.1 CMOS-, PMOS-, NMOS-, HMOS- und TTL-Bausteine 15
1.2 Struktur von Mikroprozessoren 16
1.3 Zahlendarstellung in Mikroprozessorsystemen 18
1.4 Adressierungsarten 29

2. Der SCMP II (ISP-8A/600) Mikroprozessor 32
2.1 Anschlußbelegung und elektrische Daten 32
2.2 SCMP Architektur 38
2.2.1 Akkumulator 40
2.2.2 Extension-Register 41
2.2.3 Status-Register 41
2.2.4 Programm Counter 42
2.2.5 Pointer-Register 42
2.3 Interruptbehandlung 42

3. Die Befehle des SCMP 43
3.1 Adressierungsarten des SCMP 43
3.2 Speicherbezogene Befehle 46
3.3 Inkrement- und Dekrement-Befehle 55
3.4 Immediate (unmittelbare) Befehle 56
3.5 Sprungbefehle 60
3.6 Die Pointerregister-Austausch Befehle 63
3.7 Befehle mit dem Extension-Register 66
3.8 Serielle Ein-/Ausgabe, Schiebe- und Rotationsbefehle 68
3.9 Der Verzögerungsbefehl 70
3.10 Verschiedene 1-Byte-Befehle 72
3.11 Befehlsliste geordnet nach Hex-Kode 74
3.12 Befehlsliste alphabetisch 79

4. Anwendungs- und Programmbeispiele 80
4.1 Stoppuhr 80
4.2 Geschwindigkeitsmessung 90
4.3 Digitaluhr 100
4.4 Hex-Addition und -Subtraktion 103
4.5 Dezimale Addition und Subtraktion 106
4.6 Hex/Dezimal-Wandlung und Dezimal/Hex-Wandlung 112
4.7 SCMP als Tonfrequenzgenerator 122

5. Der Mikroprozessor 6502 126
5.1 Architektur des Bausteins 127
5.2 Befehlsliste 129

6. Der Mikroprozessor 8085A 132
6.1 Architektur des Bausteins 134
6.2 Befehlsliste 134

7. Der Mikroprozessor MC6809 138
7.1 Architektur des Bausteins 139
7.2 Befehlsliste 141

8. Der Mikroprozessor Z80 143

9. Vor- und Nachteile der besprochenen Mikroprozessoren 144

10. Kurzvorstellung einiger 16-Bit-Mikroprozessoren 145
10.1 Der 8086 von Intel 145
10.2 Der Z8000 von Zilog 146
10.3 Der MC68000 von Motorola 147
10.4 Der TMS9900 von Texas Instruments 148
10.5 Anwendungsschwerpunkte für 16-Bit-Mikroprozessoren 148

Nachwort 149
Literaturverzeichnis 150
Anhang 150

Vorwort

Das vorliegende Buch wurde für die geschrieben, die in die Mikroprozessortechnik einsteigen wollen.
Es vermittelt Grundlagen die erforderlich sind und führt am Beispiel des Mikroprozessors INS8060 (SCMP II) von National Semiconductor in die Programmierung dieser Bausteine ein.
Der relativ einfache Prozessor ist dafür recht gut geeignet. Das Umsteigen auf einen anderen, leistungsfähigeren Mikroprozessor wie Z80, M6809, 8085 oder 6502 fällt leicht, wenn man zunächst das Grundsätzliche verstanden hat.
Am Schluß des Buches folgt noch eine kurze Übersicht über die populärsten 16-Bit Mikroprozessoren. Damit wird das Bild nach oben abgerundet.
Nicht behandelte Fragen wie z.B. Eigenbau eines Systems, Softwareunterstützung u.a. werden vom Verfasser gerne beantwortet.

1. Fachwortlexikon

Hier werden die im Buch verwendeten Fachwörter bzw. Begriffe vorab erklärt. Darüber hinaus sind auch Abkürzungen und Schlagworte aus der Daten-/Mikroprozessortechnik und deren Sinn enthalten.

ACC – oft verwendete Abkürzung für Akkumulator (accumulator)

Adresse – Wird die Adresse in Form eines Bitmusters auf den Adreßbus gelegt, so wird genau eine Speicherstelle (in der Regel 1 Byte) angesprochen.

Adreßbus – Über den Adreßbus werden die Adressen übertragen. Meist ist er 16 Bit breit. Oft steht er gemultiplext mit anderen Prozessorsignalen zur Verfügung.

Akkumulator – zentrales Arbeitsregister bei Mikroprozessoren über das fast alle Operationen laufen.

ALU – arithmetic and logical unit; in dieser Einheit werden alle logischen und arithmetischen Operationen ausgeführt.

AND – Englischer Ausdruck für logisches UND. Beim AND wird der Ausgang eines Logikgatters nur logisch »1«, wenn alle Eingänge des Gatters logisch »1« sind. In allen anderen Fällen ist der Ausgang »0«.

ASCII – American standard code for information interchange; Ein 7-Bit-Kode, der die Darstellung von 128 Ziffern, Groß- und Kleinbuchstaben, Sonderzeichen und Steuerzeichen erlaubt.

Assembler – Übersetzungsprogramm, welches die Umsetzung eines in mnemonischem Kode geschriebenen Programms in Maschinensprache durchführt.

BASIC – Beginners All Purpose Symbolic Instruction Code; BASIC ist eine einfach strukturierte, relativ leicht erlernbare Programmiersprache. Es gibt für sie inzwischen viele Abarten bzw. unterschiedliche Versionen. Man spricht deshalb auch von verschiedenen BASIC-Dialekten, wie z. B. NIBL-Basic (für den SCMP), TINY-Basic oder TRS80-Level II-Basic.

BCD – binary coded decimals; Die Ziffern Ø bis 9 werden mit vier Bit binär kodiert (dargestellt).

Befehlszyklus – Es ist die gesamte Zeitspanne, die zum Holen, Dekodieren und Ausführen eines Befehls vergeht.

Bit – Kleinste Informationseinheit in der Datentechnik. Ein Bit kann die logischen Zustände »0« oder »1« annehmen.

borrow – borgen; Übertrag bzw. »Anleihe« bei der Subtraktion.

Bus – System von Leitungen (Übertragungswegen) in einem Rechner oder in einem Mikroprozessorbaustein. Je nach Funktion unterscheidet man Datenbus, Adreßbus und Steuerbus.

Byte – Datenwort mit acht Bit. Alle 8-Bit Mikroprozessoren sind mehr oder weniger auf Byteverarbeitung ausgelegt.

Carry – Übertrag bei der Addition. Tritt bei der Addition zweier Binärzahlen ein Übertrag in der höchstwertigen Stelle auf, so wird im Statusregister eines Mikroprozessors das »Carry-Bit« gesetzt.

Chip – Anderer Ausdruck für eine integrierte Schaltung. Der Ausdruck kommt aus der Halbleiterherstellung, wo die integrierte Schaltung ohne Gehäuse als »chip« bezeichnet wird.

CMOS – Complementary Metal Oxid Semiconductor; Bezeichnung für eine Halbleitertechnologie, bei der P- und N-Kanal-Transistoren auf dem gleichen Substrat erzeugt werden.

condition code register – enthält die Ergebnis-Signale nach Operationen z. B. positiv, negativ, Überlauf u.a.. Es zeigt damit bestimmte Zustände innerhalb der CPU an.

CONT – »Continue«-Eingang beim Mikroprozessor SCMP. Eine logische »0« an diesem Anschluß hält den Prozessor an, wenn er die gerade aktive Instruktion abgearbeitet hat. Bei einer »1« an »CONT« arbeitet der Mikroprozessor kontinuierlich.

Control – Steuerung; Bus-Control, um nur ein Beispiel zu nennen, bedeutet Bussteuerung.

CPU – central processing unit; Zentraleinheit eines Computers. Der Mikroprozessor bildet die CPU bei mit ihm aufgebauten Systemen.

CRT – andere Bezeichnung für einen Bildschirm, wird meist im Zusammenhang mit Terminals genannt.

cross-assembler – ein Übersetzungsprogramm, das auf einem Rechner anderer Bauart läuft als auf dem, für den das Objekt-Programm geschrieben ist.

DEBUG – Bezeichnung für ein Hilfsprogramm, mit dem Adreßinhalte kontrolliert und geändert werden können. Meist lassen sich mit Hilfe des DEBUG, ein oder mehrere Unterbrechungspunkte in ein zu überprüfendes Programm setzen. Läuft der Prozessor beim Abarbeiten auf einen solchen »Breakpoint«, so werden z. B. Registerinhalte angezeigt oder ausgedruckt. Debugging heißt frei übersetzt »entwanzen«.

Digit – Zeichen oder Ziffer aus einem Zeichenvorrat.

DIL – dual in line; Bezeichnung für die übliche Form der Gehäuse von integrierten Schaltungen. Sagt aus, daß die Bausteinanschlüsse in zwei parallelen Reihen angeordnet sind.

Displacement – Wert, der zusammen mit einem Basiswert, z. B. Stand des Programmzählers, die effektive Adresse ergibt. Negative Displacements werden im Zweierkomplement dargestellt.

Display – Anzeigeeinheit in einem Mikroprozessorsystem. Meist aus mehreren Siebensegment-Anzeigebausteinen zusammengesetzt.

DMA – direct memory access; Direkter Speicherzugriff ohne Zutun des Prozessors. Es werden Zeiten, in denen der Prozessor nicht den Bus benutzt, für den Speicherzugriff ausgenutzt. Die Steuerung erfolgt mit Hilfe sogenannter DMA-Bausteine.

enable — freigeben; z.B. Unterbrechungen zulassen.

EPROM — erasable programmable ROM; Lösch- und programmierbarer Festwertspeicher.

Extensionregister — Register des SCMP, welches als Erweiterungsregister zum Akkumulator gedacht ist.

FIFO – first in first out; Methode beim Schreiben und Lesen von Daten in Speicherbereiche.

Flag – Flagge, Merker oder Signal. Mit Hilfe eines »Flags« können bestimmte Zustände angezeigt bzw. gesichert werden.

Flipflop – Schaltelement, das zwei feste logische Zustände annehmen kann (z. B. Ein/Aus, 1/0, High/Low). Man spricht auch von einer bistabilen Kippstufe.

Floppy-Disk – Magnetspeicherplatte; flexible, doppelseitig magnetisch beschichtete Kunststoffscheibe, die ca. 256 KByte Speicherkapazität besitzt.

FORTRAN – formula translation; weit verbreitete höhere Programmiersprache, die besonders für die Lösung mathematischer Probleme geeignet ist.

Gate – Schicht zwischen Drain (Senke) und Source (Quelle) bei Feldeffekttransistoren.

Hardware – Begriff für alle Bauteile und Geräte eines Systems.

Half-Carry – Einige Mikroprozessoren zeigen in ihrem Statusregister bei der Addition an, ob ein Übertrag von Bit 3 nach Bit 4 stattgefunden hat. Bei der BCD-Addition muß in diesem Fall eine Korrektur des Ergebnisses durchgeführt werden.

handshake – Quittungsbetrieb; Die Datenübertragung von einem Gerät zu einem anderen Gerät oder eine andere Schaltung, erfolgt mit Hilfe von Quittungssignalen, wie z.B. CTS (clear to send) oder TRY (transmission ready).

Hexadezimalsystem – Zahlensystem mit sechzehn unterschiedlichen Zeichen, den Ziffern Ø bis 9 und den Buchstaben A bis F.

Hex-Kode – In den Befehlslisten von Mikroprozessoren wird der binäre Instruktionskode meist Hexadezimal dargestellt, da die binäre Schreibweise zu lang und unübersichtlich ist.

HIGH – logischer Zustand; logisch »wahr« oder »1«. Kommt von hohem (high) Spannungspegel.

HMOS – high density NMOS; MOS-Technologie mit besonders feinen Strukturen und deshalb sehr hoher Integrationsdichte.

HOB – High Order Byte; höherwertiges Byte eines 16-Bit-Wortes bzw. einer Adresse.

IC – integrated circuit; als Kurzform für integrierte Schaltung verwendet.

Indexregister – Register eines Mikroprozessors über das die indizierte Adressierung abläuft, bzw. das für Zähloperationen benutzt wird.

Input – Eingabe oder Eingang.

Interface – Schnittstelle z.B. zwischen Rechner und Drucker. Ein Interface kann auch ein Baustein zur Pegelumsetzung z.B. V24 auf 20mA sein.

Interrupt – Unterbrechung; alle Mikroprozessoren besitzen einen oder mehrere Interrupteingänge, über die dem Prozessor Unterbrechungsanforderungen mitgeteilt werden können.

I/O-Port – Daten Ein-/Ausgabekanal.

IRQ – interrupt request; Eingang an vielen Mikroprozessoren über den dem Prozessor eine Interruptanforderung (interrupt request) mitgeteilt werden kann. Ist meist mittels Interruptmaske maskierbar.

IS – Abkürzung für integrierte Schaltung.

ISR – Interrupt-Service-Routine; Programm, welches die Unterbrechungsanforderung bedient.

K – steht für den Faktor 1024. Speichergrößen oder Adreßbereiche werden meist in KByte angegeben. 64KByte sind demnach 65536 Byte. Nicht zu verwechseln mit »k« für Kilo.

Keyboard – Tastenfeld; Bezeichnung für eine Tastatur, die zur Eingabe von Daten und Befehlen verwendet wird.

Label – Marke bzw. Sprungziel in einem Programm.

Latch – Zwischenspeicher, mit dem kurzzeitige Signale oder auf dem Bus anliegende Daten gespeichert werden können.

LED – light emitting diode; Leuchtdiode.

Link – Bezeichnung für eine Verbindung. Dies kann auch eine Verbindung von mehreren Unterprogrammen sein.

LIFO – last in first out; Zugriffsverfahren auf einen Speicher, meist Stapelspeicher (Stack).

Loader – Programm, welches z.B. beim Systemstart oder nach einem Reset, das Betriebssystem von einem externen Speichermedium in den Arbeitsspeicher des Rechners lädt.

LOB – Low Order Byte; niederwertiges Byte einer Adresse oder eines 16-Bit-Wortes.

LOW – logischer Zustand; logisch »unwahr« oder »0«. Kommt von niedrigem (low) Spannungspegel.

LSB – Least Significant Bit; niederwertiges Bit eines Datenwortes bzw. einer Dualzahl.

LSI – large scaled integration; Technik hoher Integrationsdichte.

memory – Speicher; allgemeiner Ausdruck für alle Speicher in einem Rechnersystem.

memory mapped I/O – Hierbei werden Ein-/Ausgänge wie Speicheradressen behandelt. Ein- oder Ausgänge können beliebige Adressen zugeordnet werden. Im Gegensatz dazu, muß der I/O bei einigen Mikroprozessoren über spezielle I/O-Adressen durchgeführt werden.

Mikroprogramm – steuert die Ausführung der Befehle in der CPU. Für jede Instruktion wird ein spezielles Mikroprogramm ausgeführt.

Mikrozyklus – Die Ausführung eines Befehls wird in einzelne Zyklen unterteilt, wie z.B. Daten ins Rechenwerk transportieren, Verknüpfungen, Ergebnis in den

Speicher schreiben u.a. Je nach Struktur eines Mikroprozessors werden mehr oder weniger Mikrozyklen pro Instruktion benötigt.

mnemonischer Befehl – In Buchstaben dargestellter Befehl eines Mikroprozessors anstelle des Maschinenkodes, z.B. LDA **lo**a**d** **a**ccu.

Monitor – a) Bezeichnung für ein Bildschirmgerät.
b) Bezeichnung für ein Hilfsprogramm, welches hauptsächlich bei kleineren Mikroprozessorsystemen vorhanden ist. Mit Hilfe des Monitors können z.B. Programme von Kassette geladen oder darauf geschrieben werden. Es können außerdem Adreßinhalte geändert oder überprüft werden. Oft sind noch verschiedene Hilfsprogramme für Zahlenumwandlung, Displacementberechnungen und spezielle Anzeigeroutinen im Monitor enthalten.

MP – manchmal verwendete Abkürzung für Mikroprozessor.

MSB – Most Significant Bit; höchstwertiges Bit eines Datenwortes.

MSI – medium scaled integration; Technik mittlerer Integrationsdichte. Schaltungen enthalten einige Hundert Transistorfunktionen je Chip.

NADS – negative address strobe; Ausgang beim Mikroprozessor SCMP, der signalisiert, wann am Systembus eine gültige Adresse anliegt.

NAND – not AND; logisches »Nicht-UND«-Gatter. Der Ausgang dieses Verknüpfungsgliedes ist nur dann »0«, wenn alle Eingänge logisch »1« sind, bzw. der Ausgang ist immer »1«, wenn einer der Eingänge »0« ist.

NBREQ – negative bus request; Anschluß des SCMP der als Ein- und Ausgang fungieren kann. Als Ausgang betrachtet, zeigt der Prozessor, daß er den Bus benutzen will. Betreibt man den Anschluß als Eingang, so wartet der Prozessor mit dem Buszugriff bis er dazu die Freigabe erhält.

NENIN – negative enable input; Eingang, der dem Mikroprozessor SCMP anzeigt, wann er auf den Systembus zugreifen darf.

NENOUT – negative enable output; Ausgang des SCMP, der signalisiert ob der Prozessor gerade auf den Systembus zugreift.

NHOLD – negative hold input; Über diesen Eingang kann der Schreib- oder Lesezyklus des SCMP so verlängert werden, daß auch langsame Speicher oder periphere Geräte mit der CPU verkehren können.

NMI – non maskable interrupt; Nicht maskierbarer Interrupteingang bei einigen Mikroprozessoren. Eine Unterbrechungsanforderung an diesem Eingang wird vom Prozessor auf jeden Fall beantwortet.

NMOS – N-Chanel Metal Oxid Semiconductors; MOS-Technologie, bei der N-Kanal Transistoren auf einem Substrat aus P-Silizium aufgebaut werden.

NOR – not OR; »Nicht-ODER«-Gatter. Der Ausgang ist nur dann logisch »1«, wenn alle Eingänge logisch »0« sind.

NRDS – negative read data strobe; Beim Lesezyklus des SCMP wird dieser Ausgang kurzzeitig logisch »0«. Mit der »0/1«-Flanke des Signals werden die Daten vom Prozessor übernommen. Hat der SCMP im Lesezyklus eine gültige Adresse auf den Adreßbus gelegt, so wird dieser Ausgang »1«. Zu allen anderen Zeiten nimmt dieser Anschluß den »Tri-State«-Zustand ein.

NRST – negative reset input; Legt man diesen Eingang des SCMP auf logisch »0«, so wird die gerade ablaufende Operation abgebrochen. Der Wechsel von »0« auf »1« setzt alle benutzbaren Register auf Null. Die nächste Instruktion wird von der Adresse 0001_{16}(!) geholt.

NWDS – negative write data strobe; Zwischen den Input-/Output-Zyklen ist dieser Ausgang im hochohmigen Zustand. Liegen gültige Daten auf dem Datenbus, so setzt der SCMP diesen Ausgang kurzzeitig auf logisch »0«.

Objektkode – Maschinenkode; Bitmuster, welches der Prozessor versteht.

Offset – Wird z.B. eine Speicherstelle über ein Basisregister adressiert, so ist die effektive Adresse, gleich Basisadresse plus/minus einem bestimmten Wert. Dieser Wert wird mit Offset oder Displacement bezeichnet.

Oktalsystem – Zahlensystem mit einem Zeichenvorrat von acht unterschiedlichen Zeichen (0, 1, 2, 3, 4, 5, 6, 7)

Operationskode – Wird oft auch mit OP-Kode abgekürzt und bezeichnet den Befehlskode, der für eine bestimmte Verarbeitung im Prozessor sorgt.

OR – logisches ODER; ODER-Gatter nehmen am Ausgang immer dann den logischen Zustand »1« an, wenn mindestens ein Eingang logisch »1« ist.

Output – Ausgabe oder Ausgang.

overflow – Überlauf; im Statusregister der meisten Mikroprozessoren gibt es ein sogenanntes »Overflow-Bit« (OV). Dieses Bit wird immer gesetzt, wenn bei arithmetischen Operationen eine Überschreitung des Zahlenbereichs auftritt.

Page – bestimmter Speicherbereich; z.B. sind beim SCMP immer 4 KByte eine »Page«.

PASCAL – Höhere Programmiersprache, die das strukturierte Programmieren besonders unterstützt.

PC – Abkürzung für »Programm Counter«. Register in einem Mikroprozessor, welches in den meisten Fällen auf die als nächste auszuführende Instruktion zeigt.

Peripherie – alle außerhalb eines Rechners liegende Geräte. Bei einem Mikroprozessorsystem kann das z.B. eine LED-Zeile sein.

Pin – Die Anschlüsse von integrierten Bausteinen werden auch als Pins bezeichnet.

PL/M – Höhere Programmiersprache für Mikroprozessoren. Von der Firma Intel in erster Linie für den 8080, 8085 und den 8086 erstellt.

PMOS – P-Chanel Metal Oxid Semiconductors; MOS-Technologie, bei der P-Kanal-Transistoren auf einem Substrat aus N-Silizium aufgebaut werden. PMOS-Bausteine sind relativ langsam und benötigen außer der positiven, noch eine negative Versorgungsspannung.

Pointerregister – Register, die in der Regel Adressen enthalten. Diese Adressen zeigen z.B. auf einen bestimmten Speicherbereich, auf eine Datentabelle oder auf einen Eingang. Mit Hilfe dieser »Pointer« kann jede beliebige Stelle im möglichen Adressierbereich angesprochen werden. Der SCMP-Mikropro-

zessor besitzt neben dem Programm Counter noch drei weitere Pointerregister.

PROM – programmable ROM; programmierbarer Festwertspeicher. Der Vorgang der Programmierung ist nicht umkehrbar.

RAM – random access memory; Schreiblesespeicher der seine Information bei Spannungsverlust verliert.

Refresh-Logik – Auffrischlogik; dynamische RAM-Speicher müssen in bestimmten Zeitintervallen (Millisekunden) aufgefrischt werden. Die durch innere Verluste verloren gegangene Ladung der einzelnen Speicherzellen wird dabei wieder erneuert.

Register – Bezeichnung für meist innerhalb eines Mikroprozessors liegende, schnelle Schreiblesespeicher, mit 8 oder 16 Bit Datenbreite.

relocatable – Programm, das aus frei im Speicher verschiebbarem Kode aufgebaut ist, d.h. es kann an beliebiger Stelle im Speicher stehen und laufen.

RESET – Rücksetzen auf einen Ausgangszustand.

ROM – read only memory; Festwertspeicher aus dem nur gelesen werden kann. Verliert seine Information nicht, wenn die Spannung abgeschaltet wird. Wird bei der Herstellung mit Hilfe von Masken programmiert.

Routine – Programme werden oft auch als Routine bezeichnet.

Sedezimalsystem – Treffendere Bezeichnung für das Hexadezimalsystem. Es handelt sich dabei um ein Zahlensystem mit 16 unterschiedlichen Zeichen (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

SENSE A, B – Fühlereingänge beim SCMP Mikroprozessor. Einer der Eingänge kann auch die Funktion eines Interrupteingangs übernehmen.

SIN – serial input; Eingang beim SCMP über den eine serielle Dateneingabe erfolgen kann. Mit einem speziellen Befehl wird die am SIN-Eingang anstehende Information in das MSB des Extensionregisters übernommen.

SOUT – serial output; Ausgang des SCMP der zur seriellen Datenausgabe benutzt werden kann. Der gleiche Befehl, der auf den seriellen Dateneingang wirkt, wirkt auch auf SOUT. Hierbei wird jedoch das LSB des Extensionregisters an SOUT ausgegeben.

Software – alle Programme werden als Software bezeichnet.

Stackpointer – Register, das auf den Stack (Stapelspeicher) zeigt.

Statusregister – Register eines Mikroprozessors, welches Informationen über den Prozessorzustand enthält. Die einzelnen Bits dieses Registers haben unterschiedliche Bedeutungen. Beispiele dafür sind »C« für Carry, »OV« für Overflow, »N« für negatives Ergebnis u.a.

String – Bezeichnung für eine Zeichenkette. Eine zusammenhängende Folge von ASCII-Zeichen wird auch als »ASCII-String« bezeichnet.

Subroutine – Unterprogramm; ein Hauptprogramm setzt sich meist aus mehreren Unterprogrammen zusammen.

Substrat – Mit Substrat wird bei Halbleiterbausteinen das Basismaterial bezeichnet, auf das die Einzelelemente aufgebaut werden.

Terminal – Bezeichnung für eine Datenein-/-ausgabe-Station. Besteht in der Regel aus Bildschirm und ASCII-Tastatur.

Tristate – Zusätzlich zu den Zuständen »0« und »1«, beide niederohmig, kommt ein dritter hochohmiger Zustand dazu. Befindet sich ein Baustein im »tristate«-Zustand, dann ist er praktisch für die anderen Komponenten nicht vorhanden.

TTL-Bausteine – In der bipolaren Halbleitertechnologie aufgebaute integrierte Schaltungen. Die Abkürzung steht für Transistor/Transistor-Logik. Hierbei sind die internen Schaltelemente aus Transistoren aufgebaut.

TTL-Pegel – TTL-Bausteine interpretieren alle Spannungen die zwischen Ø V und + 0,8 V liegen als logische »0«. Spannungen zwischen + 2 V und Versorgungsspannung (+ 5 V) haben den logischen Zustand »1«.

VLSI – very large scaled integration; Bezeichnung für sehr hohe Integrationsdichte. Bausteine dieser Herstellungsform vereinigen mehrere Zehntausend Transistorfunktionen auf einem Chip.

worst case – Der ungünstigste aller Fälle. Man kann z. B. die Laufzeit eines verschachtelten Programms für den »worst case« angeben. Es ist die Zeit, die benötigt wird, wenn alle ungünstigen Bedingungen, wie z. B. maximale Schleifendurchläufe, größtmögliche Zahlengröße u. a. zusammenkommen.

XOUT – Taktfrequenzausgang beim Mikroprozessor SCMP.

Zeitmultiplex – bedeutet zeitlich hintereinander. Werden z. B. die Zustände eines Mehrfacheingangs nacheinander abgefragt und übertragen, so spricht man von einer (zeit-)multiplexen Übertragung.

1.1 CMOS-, PMOS-, NMOS-, HMOS- und TTL-Bausteine

Eigentlich begann es 1948, als der Transistor von den Amerikanern J. Bardeen, W.H. Brattain und W. Shockley erfunden wurde.
Der nächste Schritt waren die ersten Transistor-Arrays. Mit zunehmendem »know how« in der Halbleiterindustrie gelang es, immer mehr Einzelkomponenten auf einem »Chip« zu integrieren. Von den MSI (medium scaled integration) Schaltungen mit einigen Hundert Transistorfunktionen in einem Baustein ging die Entwicklung weiter zu den LSI-IC's. Hier sind schon einige Tausend Funktionen auf einem Schaltkreis vereinigt. Die neueste Entwicklung nach der LSI (large scaled integration)-Technik sind Bausteine in VLSI (very large scaled integration)-Technologie. Integrierte Schaltungen dieser Generation enthalten einige Zehntausend einzelne Transistorfunktionen auf einem Chip.
Um die chronologische Reihenfolge zu wahren, wenden wir uns zunächst den **TTL-Bausteinen** zu. Sie sind in bipolarer Technik hergestellt. Bipolar deshalb, weil hier gleichzeitig positive und negative Ladungsträger am Ladungstransport beteiligt sind. Die integrierten Transistorfunktionen sind im Prinzip aufgebaut wie herkömmliche bipolare Transistoren.
TTL-Bausteine zeichnen sich durch hohe Arbeitsgeschwindigkeiten (30–100 ns) aus. Nachteilig ist ihre relativ hohe Verlustleistung und die niedrige Integrationsdichte (bis 500 Transistorfunktionen pro Chip).
Im Gegensatz zu den TTL-Schaltkreisen sind die MOS (metal oxide semiconductor)-Bausteine aus Feldeffekt-Transistoren (FET) aufgebaut. Sie zeichnen sich durch sehr niedrige Verlustleistung aus. Der Platzbedarf eines MOS-Transistors beträgt nur etwa ein Zehntel eines bipolaren Transistors.
Bei der **PMOS-Technik** wird als Substrat N-Silizium verwendet. Die Transistoren sind selbstsperrende P-Kanal-Typen. In PMOS-Technik hergestellte Bausteine spielen heute eine untergeordnete Rolle, da sie relativ langsam sind und zusätzlich eine negative Betriebsspannung benötigen.
Im Gegensatz dazu sind Bausteine in **NMOS-Technologie** fast genauso schnell wie bipolare Bausteine, benötigen nur eine Versorgungsspannung und sind zudem noch TTL-kompatibel (Logikpegel).
Die N-Kanal Transistoren werden auf einem Substrat aus P-Silizium aufgebaut.
In der **CMOS-Technik** werden N- und P-Kanal Transistoren auf dem gleichen Substrat erzeugt. Dies erfordert jedoch zusätzliche Arbeitsschritte bei der Herstellung und macht somit die Bausteine etwas teurer als die NMOS-IC's. Die Integrationsdichte beträgt nur etwa ein Zehntel der NMOS-Technik. Dafür ist der Leistungsbedarf um den Faktor fünf geringer. Hauptvorteile sind die recht hohe Arbeitsgeschwindigkeit (wie Standard TTL), Betrieb in einem großen Speisespannungsbereich (3–15 V) mit einem Störspannungsabstand von 45% der Versorgungsspannung. Bei Betrieb an 5V sind CMOS-Bausteine TTL-kompatibel.
Die MOS-Technik ist mittlerweile bei HMOS (high density NMOS) angelangt. Durch immer kleinere Gatelängen, man ist heute bei der HMOS[1]-Technik bei ca. 3,5 Mikrome-

ter, können immer mehr einzelne Transistorfunktionen bei gleichzeitig höherer Arbeitsgeschwindigkeit auf einem Chip integriert werden. Die Mikroprozessoren 8086 und 68000 sind in dieser Technologie hergestellt.
Die Halbleiterhersteller streben für die nächsten Jahre HMOS^2 mit 2 Mikrometern Gatelänge und HMOS^3 mit weniger als 1 Mikrometer Gatelänge an. Die Integrationsdichte ließe sich dadurch noch einmal um den Faktor drei steigern.
Der weiter interessierte Leser sei auf entsprechende Fachliteratur und Firmenschriften hingewiesen. Als Suchbegriffe können ihm z. B. SOS, I^{2}L, XMOS und VMOS dienen.
Allen gemeinsam ist, daß Logikbausteine nur zwei gültige Pegelbereiche haben. Den Zustand »HIGH«, der dem logischen »wahr« entspricht und der Zustand »LOW«, der dem logischen »unwahr« entspricht. Der bei den Bausteinen mit »tristate« Ausgängen mögliche dritte Zustand ist kein Logikpegel.

.2 Struktur von Mikroprozessoren

ür den Laien stellt sich oft die Frage: »Was verbirgt sich im Innern eines Mikroprozessor-Chips, damit dieser Baustein als Prozessor arbeitet?«
Es muß, wie leicht einzusehen ist, eine »Steuereinheit« enthalten sein. Weiterhin ist zur Verarbeitung von arithmetischen und logischen Befehlen eine »Recheneinheit«, auch mit ALU (**a**rithmetic and **l**ogic **u**nit) bezeichnet, enthalten. Ein wichtiger Funktionsblock ist die »Befehlsdekodiereinheit«. Für den Informationsaustausch mit der »Außenwelt« (Speicher, Ein-/Ausgabegerät usw.) über den Adreß-, Daten- und Steuerbus enthält unser Mikroprozessor eine »Ein-/Ausgabe-Steuereinheit«. Zur internen Datenverarbeitung sind außerdem mehrere »Register« auf dem Chip integriert.
Ein System im Mikroprozessor, ohne das er nicht arbeiten kann, ist das »interne Bussystem«. Es besteht in der Regel aus Adreßbus, Datenbus und Steuerbus, welche die einzelnen Komponenten miteinander verbinden. Bei einigen Prozessoren wird ein sogenannter Multiplexbus verwendet. Er enthält abhängig von bestimmten Steuersignalen Daten, Steuer- oder Statussignale oder Adressen.
Die modernen Mikroprozessoren haben eine integrierte Takterzeugung, d. h. der Taktgenerator, welcher den Systemtakt liefert, ist im Schaltkreis integriert.
Mikroprozessoren, die mikroprogrammiert sind (es gibt auch andere) enthalten zusätzlich noch den »Mikroprogrammspeicher«, der das sogenannte Mikroprogramm enthält. Dieses Programm sagt der Steuereinheit, was wie mit wem getan werden soll, d. h. jeder einzelne Befehl wird als kleines Programm ausgeführt.
Bild 1.2.1 zeigt den prinzipiellen Aufbau eines Mikroprozessors. Auf die unterschiedlichen Strukturen einzelner Bausteine wird in Kapitel 2 und 5 eingegangen.

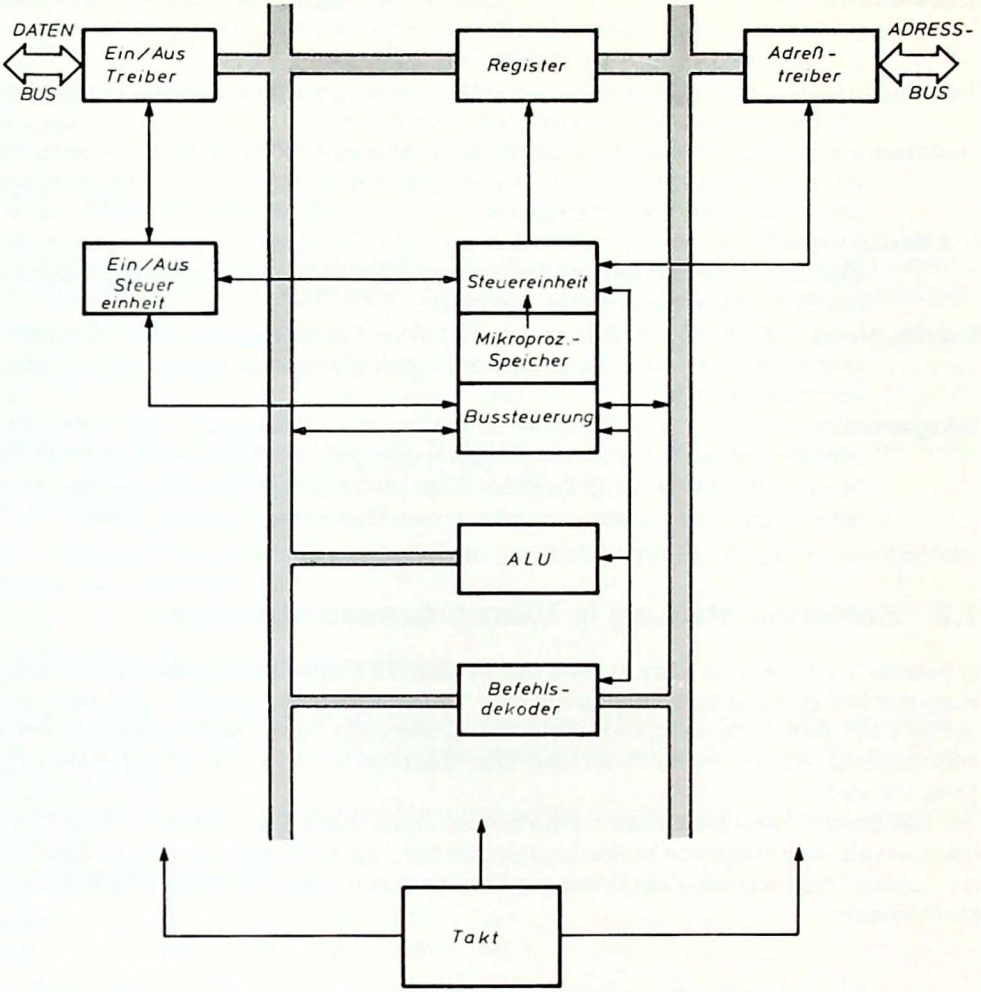


Bild 1.2.1: Prinzipielle Struktur eines Mikroprozessors

Einzelkomponenten:

ALU – (arithmetic logic unit), Recheneinheit. Hier werden alle logischen und arithmetischen Instruktionen ausgeführt.

Steuereinheit – steuert den gesamten Ablauf, meist selbst wiederum vom Mikroprogramm gesteuert. Veranlaßt z. B. das Holen eines Wertes aus dem Speicher und dessen Weitertransport in die ALU.

Befehlsdekoder – er entschlüsselt den Befehlskode und veranlaßt die Steuereinheit, ein bestimmtes Mikroprogramm auszuführen.

Registersatz – dieser enthält mindestens ein zentrales Arbeitsregister, den Akkumulator. Hinzu kommen noch der Programmschrittzähler, ein Statusregister und meist mehrere Hilfsregister.

E/A-Steuereinheit – sie sorgt dafür, daß z. B. die Eingänge in den hochohmigen Zustand (tri-state) geschaltet werden oder daß Daten vom Datenbus über die E/A-Treiber nach außen gelangen.

Bussteuerung – die zeitliche Zuordnung einzelner Funktionsgruppen zu Datenbus und Adreßbus werden von dieser Einheit gemanagt. Angestoßen wird sie vom Steuerwerk.

Taktgenerator – er liefert den internen Einheiten den Arbeitstakt. Das kann unter Umständen ein aus dem von außen angelegten einfachen Takt abgeleiteter Mehrphasentakt sein. Oft werden über Treiberausgänge auch ein oder mehrere Taktsignale an die Anschlüsse des Mikroprozessors geliefert.

1.3 Zahlendarstellung in Mikroprozessorsystemen

In Kapitel 1.1 haben wir den Aufbau der heutigen integrierten Bausteine und damit auch der Mikroprozessoren kennengelernt. Dabei haben wir gesehen, daß ein Logikgatter zwei definierte Ausgangszustände einnehmen kann. Den Zustand »HIGH« nachfolgend mit »1« bezeichnet und den Zustand »LOW«, der im Weiteren »0« genannt wird.
Der Mikroprozessor verarbeitet nur die beiden Pegel 0 und 1, d. h. Zahlen müssen dem Baustein als eine Folge von Nullen und Einsen bereitgestellt werden. Diese Folge wird als Dualzahl bezeichnet. Zum besseren Verständnis machen wir einen Ausflug in die Mathematik.

1.3.1 Das Dezimalsystem

Im täglichen Leben benutzen wir für unsere Zahlendarstellungen das Zehnersystem oder besser Dezimalsystem. Es handelt sich dabei um ein Stellenwertsystem mit einem Vorrat von 10 unterschiedlichen Zeichen (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Die Wertigkeit der einzelnen Zeichen ist abhängig von ihrer Stelle in der Zeichenfolge.

Es gilt:

Stelle	5	4	3	2	1
Wert	10^4	10^3	10^2	10^1	10^0

Die Zahl 81327 kann demnach auch als Folge

$8 \cdot 10^4 + 1 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0$

geschrieben werden. Ihr Wert ist in beiden Fällen gleich. Die Basis dieses Systems ist die »Zehn« (10).

1.3.2 Das Dualsystem

Im weiter oben bereits angesprochenen Dualsystem gibt es nur zwei unterschiedliche Zeichen, die 0 und die 1. Auch beim Dualsystem handelt es sich um ein Stellenwertsystem. Hier gilt:

Stelle	5	4	3	2	1
Wert	2^4	2^3	2^2	2^1	2^0

Nehmen wir eine beliebige Dualzahl 1001101, so läßt sich diese auch in der Form $1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$ schreiben.
Der Wert ist 77_{10}.

Wir sehen, daß man selbst für kleine Zahlen bei der Darstellung im Dualsystem schon relativ viele Stellen braucht.

1.3.3 Das Sedezimalsystem

Eine kürzere Schreibweise erhält man mit dem nachfolgend beschriebenen Sedezimalsystem, manchmal auch mit Hexadezimalsystem bezeichnet. Die Basis dieses Systems ist 16.
Das Sedezimalsystem ist, wie wir später noch sehen werden, eng mit dem Dualsystem verwandt.
Für die Stellenwertigkeit gilt:

Stelle	5	4	3	2	1
Wert	16^4	16^3	16^2	16^1	16^0

Zur Zahlendarstellung stehen 16 unterschiedliche Zeichen zur Verfügung (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Da beim Arbeiten mit dem Mikroprozessor praktisch nur das Sedezimalsystem verwendet wird, sollte man sich schon jetzt die Werte der Sedezimalzeichen einprägen.

Sedezimalzeichen:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Wert im Dez.-Syst:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Die Zahl $2A7_{16}$ kann demnach auch als Folge

$$2 \cdot 16^2 + 10 \cdot 16^1 + 7 \cdot 16^0$$

geschrieben werden. Der dezimale Wert ist 423.

1.3.4 Das Oktalsystem

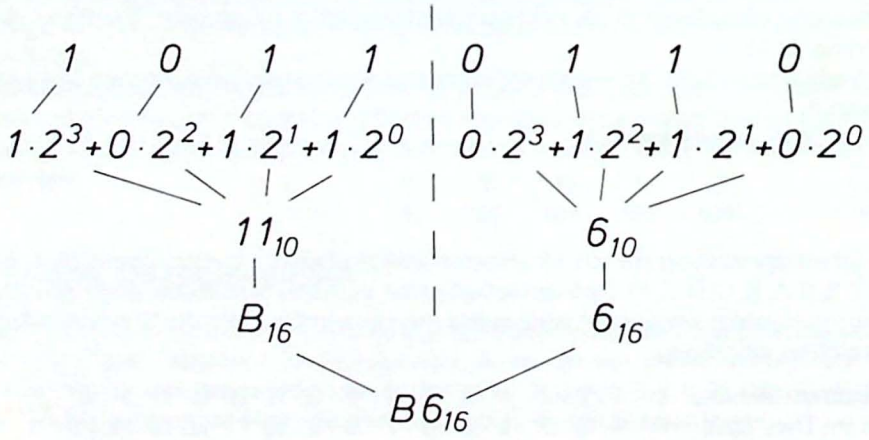
Auch das Oktalsystem ist ein Stellenwertsystem. Diesmal zur Basis 8. Es wird in Mikroprozessorsystemen jedoch recht selten verwendet, obwohl es leichter zu handhaben ist als das Sedezimalsystem. Hier stehen acht unterschiedliche Zeichen zur Zahlendarstellung zur Verfügung (0, 1, 2, 3, 4, 5, 6, 7).

Stelle	5	4	3	2	1
Wert	8^4	8^3	8^2	8^1	8^0

Ebenso wie bei den bereits besprochenen Zahlensystemen können wir auch hier eine beliebige Oktalzahl 731_8 als Folge darstellen.
Sie lautet $7 \cdot 8^2 + 3 \cdot 8^1 + 1 \cdot 8^0$ und hat den Wert 89_{10}.

1.3.5 Umwandlung von einem Zahlensystem in ein anderes

Im Weiteren wollen wir uns noch mit dem Dual- und Sedezimalsystem beschäftigen. Wie bereits gesagt, sind die beiden Zahlensysteme eng miteinander verwandt. Nehmen wir eine beliebige Dualzahl 10110110. Um nun von der binären zur sedezimalen Darstellung zu kommen unterteilen wir die Dualzahl in Gedanken von rechts beginnend in Gruppen je vier Bit auf. Jetzt bestimmen wir den Wert dieser Gruppen wie in Abschnitt 1.3.2 gezeigt.



Diese Darstellung ist wesentlich kürzer. Einige werden sich jetzt fragen, ob sich der Wert geändert hat oder ob er gleich geblieben ist. Er ist noch gleich, wie wir sehen werden. Doch nehmen wir unsere Dualzahl und bestimmen nach den bekannten Regeln ihren Wert.

$10110110_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 182_{10}$

Das Gleiche tun wir jetzt für die oben ermittelte Sedezimalzahl.

$B6_{16} = 11 \cdot 16^1 + 6 \cdot 16^0 = 182_{10}$

Wir stellen fest: $B6_{16} = 10110110_2 = 182_{10}$

In der Praxis kommt es immer wieder vor, daß man Zahlen von einem Zahlensystem in ein anderes Zahlensystem umwandeln muß.
Wie man vom Dualsystem bzw. Sedezimalsystem ins Dezimalsystem kommt, haben wir bereits gesehen. Jetzt wollen wir uns mit der Umwandlung von Dezimalzahlen in Sedezimal- bzw. Dualzahlen beschäftigen. Es gibt verschiedene Methoden. Wir wollen uns zwei relativ einfache Verfahren anschauen. So hat jeder die Möglichkeit, den für sich angenehmsten Weg zu wählen. Bei beiden Lösungswegen kommt man ums Rechnen nicht herum.

Verfahren 1:
Hierbei wenden wir das, was wir im Dezimalsystem fast schon unbewußt tun, auf das Sedezimalsystem an. Machen wir uns doch einmal klar, wie wir Dezimalzahlen darstellen. Nehmen wir z.B. die Zahl 5374_{10}. Ihre Darstellung im Dezimalsystem lautet

$5 \cdot 10^3 + 3 \cdot 10^2 + 7 \cdot 10^1 + 4 \cdot 10^0 =$
$5000 \quad + 300 \quad + 70 \quad + 4 \quad = 5374$

Die Zahl 5374_{10} können wir fünf-(5)-mal durch 10^3 (= 1000) teilen, der Rest 374_{10} läßt sich drei-(3)-mal durch 10^2 (= 100) teilen. Der hierbei entstehende Rest 74_{10} kann sieben-(7)-mal durch 10^1 (= 10) geteilt werden; der jetzt noch übrig bleibende Zahlenwert 4_{10} kann genau vier-(4)-mal durch 10^0 (= 1) geteilt werden. Durch fortgesetztes Dividieren durch die nächst niedrigere Zehnerpotenz kann somit eine Zahl in ihre einzelnen Stellen zerlegt werden.
Genauso einfach, jedoch ungewohnt, ist dieses Verfahren im Sedezimalsystem. Bedingt durch die andere Stellenwertigkeit wird jetzt jedoch nicht durch 10^3, 10^2, 10^1 und 10^0 dividiert. Entsprechend der Stellenwertigkeit teilen wir durch 16^3 (= 4096), 16^2 (= 256), 16^1 (= 16) und 16^0 (= 1). Auf diese Weise können beliebig große Zahlen umgewandelt werden. Man muß sich nur jeweils die höchste in der Zahl enthaltene Sechzehnerpotenz bestimmen und mit dieser dann die Divisionsreihe beginnen.
Für die Zahl 5374_{10} wollen wir diese Rechnung jetzt durchführen. Zuerst stellen wir also die höchste Sechzehnerpotenz fest. Damit erhalten wir auch die höchste Stelle des Ergebnisses. Zur Erleichterung sind im Anhang Tabellen mit Potenzen zur Basis sechzehn und zwei enthalten.

Wir sehen, $16^3 = 4096_{10}$ ist die höchste Potenz von 16, die in 5374_{10} noch enthalten ist. Daraus folgt, das erste Ergebnis ergibt die vierte Stelle der Sedezimalzahl.
Doch rechnen wir.

	4.	3.	2.	1.	Stelle
$5374_{10} : 16^3 = 5374 : 4096 = \mathbf{1}$ (Rest 1278)	**1**				
$1278_{10} : 16^2 = 1278 : 256 = \mathbf{4}$ (Rest 254)		**4**			
$254_{10} : 16^1 = 254 : 16 = 15 = \mathbf{F}$ (Rest 14)			**F**		
$14_{10} : 16^0 = 14 : 1 = 14 = \mathbf{E}$ (Rest 0)				**E**	
	1	4	F	E	

Auf relativ einfache Weise sind wir von der Dezimalzahl 5374 zur Sedezimalzahl 14FE gekommen.

Zur Vertiefung gleich noch ein Beispiel. Wir wollen 65535_{10} in sedezimaler Schreibweise darstellen.

$65535_{10} : 16^3 = 65535 : 4096 = 15_{10} = F_{16}$ (Rest 4095)	F
$4095_{10} : 16^2 = 4095 : 256 = 15_{10} = F_{16}$ (Rest 255)	F
$255_{10} : 16^1 = 255 : 16 = 15_{10} = F_{16}$ (Rest 15)	F
$15_{10} : 16^0 = 15 : 1 = 15_{10} = F_{16}$ (Rest 0)	F
	FFFF

$65535_{10} = FFFF_{16} = 1111111111111111_2$

Das gleiche Verfahren wenden wir jetzt auf die Umwandlung der Dezimalzahl 278 in eine Dualzahl an. Wir müssen nun durch Potenzen zur Basis zwei dividieren. Die höchste Potenz von zwei die gerade noch in 278_{10} enthalten ist, ist die Potenz $2^8 = 256_{10}$.

Rechnen wir das Beispiel einmal durch.

$278_{10} : 2^8 = 278 : 256 = 1$ (Rest 22)	1
$22_{10} : 2^7 = 22 : 128 = 0$ (Rest 22)	0
$22_{10} : 2^6 = 22 : 64 = 0$ (Rest 22)	0
$22_{10} : 2^5 = 22 : 32 = 0$ (Rest 22)	0
$22_{10} : 2^4 = 22 : 16 = 1$ (Rest 6)	1
$6_{10} : 2^3 = 6 : 8 = 0$ (Rest 6)	0
$6_{10} : 2^2 = 6 : 4 = 1$ (Rest 2)	1
$2_{10} : 2^1 = 2 : 2 = 1$ (Rest 0)	1
$0_{10} : 2^0 = 0 : 1 = 0$ (Rest 0)	0
	100010110

$278_{10} = 100010110_2 = 116_{16}$

Auch hier ein zweites Beispiel. Die Dezimalzahl 33 soll in das Dualsystem umgerechnet werden.

$33_{10} : 2^5 = 33 : 32 = 1$ (Rest 1)
$1_{10} : 2^4 = 1 : 16 = 0$ (Rest 1)
$1_{10} : 2^3 = 1 : 8 = 0$ (Rest 1)
$1_{10} : 2^2 = 1 : 4 = 0$ (Rest 1)
$1_{10} : 2^1 = 1 : 2 = 0$ (Rest 1)
$1_{10} : 2^0 = 1 : 1 = 1$ (Rest 0)

1
0
0
0
0
1

1 0 0 0 0 1

$33_{10} = 100001_2 = 21_{16}$

Mit etwas Übung geht diese Art der Umwandlung recht flott. Das Verfahren eignet sich für jedes beliebige Stellenwertsystem.

Verfahren 2:
Bei diesem Verfahren wird die umzuwandelnde Dezimalzahl fortwährend durch die entsprechende Basiszahl des neuen Systems geteilt bis als Ergebnis Null erscheint. Die Reste der Divisionen werden angeschrieben. Alle Reste von unten nach oben gelesen stellen die Zahl im entsprechenden neuen System dar.
Hier soll uns je ein Beispiel für die Umwandlung Dezimal/Sedezimal und Dezimal/Dual genügen.
Zunächst die Dezimal-/Sedezimalwandlung

$1023_{10} : 16_{10} = 63_{10}$ Rest $15_{10} = F_{16}$ ▲
$63_{10} : 16_{10} = 3_{10}$ Rest $15_{10} = F_{16}$ |
$3_{10} : 16_{10} = 0$ Rest $3_{10} = 3_{16}$ |

Der Rest in Pfeilrichtung gelesen liefert das Ergebnis. $1023_{10} = 3\ FF_{16}$

Jetzt ein Beispiel für die Dezimal-/Dualumwandlung.

$328_{10} : 2_{10} = 164_{10}$ Rest 0 ▲
$164_{10} : 2_{10} = 82_{10}$ Rest 0 |
$82_{10} : 2_{10} = 41_{10}$ Rest 0 |
$41_{10} : 2_{10} = 20_{10}$ Rest 1 |
$20_{10} : 2_{10} = 10_{10}$ Rest 0 |
$10_{10} : 2_{10} = 5_{10}$ Rest 0 |
$5_{10} : 2_{10} = 2_{10}$ Rest 1 |
$2_{10} : 2_{10} = 1_{10}$ Rest 0 |
$1_{10} : 2_{10} = 0_{10}$ Rest 1 |

Auch hier liefern die Reste in Pfeilrichtung gelesen das Ergebnis. $328_{10} = 101001000_2$
Nachdem wir nun von einem Zahlensystem in ein anderes Zahlensystem umrechnen können, wollen wir im Dualsystem und im Sedezimalsystem rechnen.

1.3.6 Rechnen im Dualsystem und im Sedezimalsystem

Rechnen fällt den meisten Mikroprozessoren schwer. Sie sind in der Regel nicht dafür gebaut. Mit einigem Programmieraufwand aber kann ein Mikroprozessor sogar multiplizieren und dividieren. Mit sehr viel Programmieraufwand kann man ihn sogar dazu bringen, Wurzeln zu ziehen, zu potenzieren und Winkelfunktionen zu berechnen. Zum Programmieraufwand kommt noch der recht hohe Zeitbedarf hinzu. Wie schon gesagt, ist der Softwareaufwand beachtlich, wenn man mit dem Mikroprozessor z. B. Wurzeln ziehen will. Der Mikroprozessor kann »nur« addieren und subtrahieren, wobei das Subtrahieren, wie wir noch sehen werden, auch nur eine Addition ist.

Wir wissen, daß unser Mikroprozessor nur Dualzahlen oder allgemeiner gesagt, Bitmuster verarbeiten kann. Die Bedeutung dieser Muster können wir selbst (in Grenzen) festlegen.

Nachfolgende Definitionen werden nur gemacht, um das Prinzip beim Rechnen mit Dualzahlen zu zeigen. Bedingt durch die geringe Wortbreite von nur acht Bit, sind solche Zahlen wenig für den praktischen Gebrauch geeignet.

a) Positive ganze Zahlen werden durch ein acht Bit breites Datenwort, dessen höchstwertiges Bit immer Null ist, dargestellt.

b) Negative ganze Zahlen werden im Zweierkomplement dargestellt.

Die Datenwortbreite ist ebenfalls acht Bit. Das höchstwertige Bit ist immer Eins.

Bei einer solchen Darstellung kann man einen Zahlenbereich zwischen -128_{10} und $+127_{10}$ abdecken.

Ohne auf die mathematischen Hintergründe einzugehen, wollen wir uns eine sehr einfache und elegante Methode zur Zweierkomplementbildung anschauen.

Wollen wir das Zweierkomplement einer Dualzahl bilden, so müssen wir alle Bits komplementieren (aus 1 wird 0, aus 0 wird 1) und zum Ergebnis eine 1_2 addieren. Anders ausgedrückt, wir übernehmen von rechts beginnend bis einschließlich der ersten »1« die Zahl unverändert, ab dieser Stelle werden dann alle Bits komplementiert.

Das Zweierkomplement $\overline{D}$ einer n-stelligen Dualzahl D ist:

$$\overline{D} = 2^{n+1} - D$$

In den folgenden vier Beispielen steht jeweils oben die Dualzahl und darunter ihr Zweierkomplement.

1)	2)	3)	4)
01001101	00111011	01000000	00110100
10110011	11000101	11000000	11001100

Beim Rechnen wollen wir uns auf die Addition und Subtraktion beschränken.

Binäre Addition

Es gilt:

0	0	1	1
+0	+1	+0	+1
0	1	1	**10**

Übertrag in die 2^1-te Stelle.

Wir sehen, wie bei der dezimalen Addition, tritt auch bei der binären Addition ein Übertrag auf. Doch addieren wir einmal zwei Zahlen. Auf der linken Seite zur Kontrolle dezimal und auf der rechten Seite binär.

12	00001100
+47	+00101111
59	00111011

Ein weiteres Beispiel:

93	01011101
+34	+00100010
127	01111111

Addition mit Überschreitung des gültigen Zahlenbereiches.

93	01011101
+40	+00101000
133	**1**0000101
	Überlauf

Tritt bei der Addition zweier positiver Dualzahlen ein Übertrag in das höchstwertige Bit auf, so erhält man das Ergebnis indem man dieses Bit zu Null setzt und zur restlichen Dualzahl 1_2 addiert. Die dabei entstehende Zahl gibt an, um wieviel das Ergebnis grösser als 127_{10} ist. Dargestellt werden kann das Ergebnis obiger Rechnung in unserem gewählten Zahlenbereich nicht.

Binäre Subtraktion

Es gilt:

			Anleihe
0	1	1	**1**0
−0	−0	−1	−1
0	1	0	1

Im rechten Beispiel wird eine »Anleihe« bei der nächsthöheren Zweierpotenz gemacht. Beim Ergebnis handelt es sich in dem Fall um eine negative Zahl ($-1 \cdot 2^0$). Weiter oben wurde bereits gesagt, daß die Subtraktion auf eine Addition zurückgeführt wird, bzw. zurückgeführt werden kann. Dazu nachfolgende Regel.

Die Subtraktion von dual n-stellig dargestellten Zahlen kann man durch die Addition des Komplements zur 2^{n+1}-ten Potenz ersetzen. Das Ergebnis muß man an der zur 2^{n+1}-ten Potenz gehörenden Stelle um eins vermindern.

Angewendet auf unseren gewählten Zahlenbereich heißt das, daß wir zum Minuenden (Zahl, von der abgezogen wird) das Zweierkomplement des Subtrahenden (Zahl, die abgezogen wird) addieren müssen.

Das Ergebnis wird an der Stelle der Potenz 2^8 um 1 vermindert, d. h. die bei der Addition entstehende neunte Stelle wird einfach weggelassen.

Dazu gleich zwei Beispiele.

12	00001100	00001100	
− 5	− 00000101	+ 11111011	Addition d. 2er Kompl.
7		100000111	

Die unterstrichene Stelle wird weggelassen, das Ergebnis ist 7_{10}.

67	01000011	01000011	
− 33	− 00100001	+ 11011111	Addition d. 2er Kompl.
34		100100010	

Auch hier wird die unterstrichene Stelle weggelassen.

Die beiden Beispiele haben gezeigt, worauf es ankommt. Zum besseren Verständnis ist jeweils links die Rechnung dezimal angeschrieben. In der Mitte stehen die Dualzahlen und rechts wird die eigentliche Rechnung ausgeführt.

Ist das Ergebnis negativ, so hat das höchstwertige Bit den Wert »1«.

Auch dazu ein Beispiel.

12	00001100	00001100	
− 15	00001111	+ 11110001	Addition d. 2er Kompl.
− 3		11111101	

Zur Kontrolle kann das Zweierkomplement vom Ergebnis gebildet werden, es ist 3_{10}. Wenden wir uns nun dem Rechnen mit sedezimal dargestellten Zahlen zu. In kleinen Mikroprozessorsystemen erfolgt die Kommunikation mit dem Prozessor fast ausschließlich sedezimal. Darum ist es sehr wichtig, daß man sedezimal addieren und subtrahieren kann.

Sedezimale Addition

Die gleichen Regeln wie beim Rechnen im Dezimalsystem haben auch im Sedezimalsystem ihre Gültigkeit. Man muß sich jedoch immer darüber im Klaren sein, daß man es mit dem Sechzehner- und nicht mit dem Zehnersystem zu tun hat. Addieren wir doch gleich einmal zwei Sedezimalzahlen.

	$1A9_{16}$	entspricht	425_{10}
+	353_{16}		+ 851_{10}
	$4FC_{16}$		1276_{10}

Jetzt ein Beispiel mit Übertrag. Er ist in einer extra Zeile aufgeführt.

	$6D1_{16}$	entspricht	1745_{10}
+	$A5F_{16}$		+ 2655_{10}
Übertrag	111		111
	1130_{16}		4400_{10}

Das Ganze ist relativ einfach und sicher, wenn man beim Übertrag aufpaßt. Immer, wenn bei der Addition von Sedezimalzahlen die Zahl 15_{10} überschritten wird, tritt ein Übertrag auf.

Sedezimale Subtraktion

Leider kommt beim Programmieren das Subtrahieren viel häufiger vor als das Addieren. Deshalb wollen wir uns damit ausführlicher beschäftigen. Zunächst ein einfaches Beispiel. Zur Kontrolle ist rechts die Rechnung im Dezimalsystem angeschrieben.

	15_{16}	entspricht	21_{10}
−	3_{16}		− 3_{10}
	12_{16}		18_{10}

Wie bei der dezimalen Subtraktion, wird auch im Sedezimalsystem, die Rechnung für jede Stelle ausgeführt. Im obigen Beispiel lautet demnach die erste Frage: sedezimal drei und wieviel ergibt sedezimal fünf? Es ist die sedezimale Zwei. Da der Subtrahend (3_{16}) in diesem Fall kleiner als der Minuend (5_{15}) war, entsteht kein Übertrag. Die nächsthöhere Stelle des Subtrahenden ist null. Ziehen wir diese Null von der Eins des Minuenden ab, so erhalten wie sedezimal eins. Das Ergebnis lautet 12_{16}.
Zur Vertiefung gleich ein weiteres Beispiel.

	27_{16}	entspricht	39_{10}
−	8_{16}		− 8_{10}
	$1F_{16}$		31_{10}

Ganz so einfach wie im obigen Beispiel geht die Rechnung hier nicht. Da der Subtrahend (8_{16}) an der niederwertigsten Stelle größer als der Minuend (7_{16}) ist, müssen wir

uns von der nächsthöheren Stelle des Minuenden (2_{16}) eine »Anleihe« holen. Wir »borgen« uns eine Stelle. Unsere Frage für die niederwertigste Stelle lautet nun: 8_{16} und wieviel ist 17_{16}? Die Antwort ist F_{16}.
Genau wie im Dezimalsystem berücksichtigen wir die »Anleihe« in der nächsthöheren Stelle des Subtrahenden. Sehen wir uns das in unserem Beispiel an,

$$\begin{array}{rr} & 27_{16} \\ - & 8_{16} \\ \text{Anleihe} & 1 \\ \hline & 1F_{16} \end{array}$$

so wird klar, wie dieses »Borgen« geschieht. Wir müssen uns jedoch merken, daß das Leihen den Minuenden scheinbar um einen »16er« größer macht. Beim Dezimalsystem »leihen« wir uns einen »10er«.
Wir können uns demnach merken:
Ist bei einer Stelle der Subtrahend größer als der Minuend, so wird von der nächsthöheren Stelle des Minuenden ein »Sechzehner« »geliehen«. Jetzt ist die Subtraktion möglich. Subtrahiert man jetzt die nächsthöhere Stelle, dann muß man die »Anleihe« berücksichtigen.
Nehmen wir ein Beispiel aus der Praxis. Wir wollen die Differenz zweier sedezimal geschriebener Speicheradressen bilden. Im Beispiel ist auch hier rechts die Rechnung im Dezimalsystem aufgeführt.

Adresse 1	$C259_{16}$	entspricht	49753_{10}
Adresse 2	$-\ A51F_{16}$		$-\ 42271_{10}$
Differenz	$1D3A_{16}$		7482_{10}

Ein anderer, jedoch genauso oft vorkommender Fall ist, daß eine Subtraktion durchgeführt werden muß, bei der der Subtrahend größer als der Minuend ist (z. B. Rückwärtssprünge im Programm).
Dazu auch gleich zwei Beispiele.

1)	$5A2_{16}$	entspricht	1442_{10}
	$-\ 5BE_{16}$		$-\ 1470_{10}$
	$FE4_{16}$		$-\ 28_{10}$
2)	$1CA8_{16}$	entspricht	7336_{10}
	$-\ 2F18_{16}$		$-\ 12056_{10}$
	$\mathbf{F}ED90_{16}$		$-\ 4720_{10}$

Ist bei der Subtraktion zweier sedezimaler Zahlen der Subtrahend größer als der Minuend, so erscheint das Ergebnis als Komplement zur nächsthöheren Potenz von sechzehn. Man kann dies daran erkennen, daß die höchstwertige Stelle des Ergebnisses größer als 7_{16} ist.

1.4 Adressierungsarten

Nachdem wir uns die ganze Zeit mehr oder weniger intensiv mit hardwarebezogenen Problemen befaßt haben, wollen wir uns jetzt etwas der Software zuwenden.
Es gibt grundsätzlich mehrere Möglichkeiten, eine Speicherstelle oder einen Ein-/Ausgabekanal anzusprechen. Allen gemeinsam ist, daß jede Speicherstelle bzw. jeder Ein-/Ausgabekanal eine bestimmte Adresse hat. In Mikroprozessorsystemen ist ein byteorientierter Speicheraufbau üblich. Hier wird beim Anlegen einer gültigen Adresse (richtige Schreib-/Lesesignale sind vorausgesetzt) genau ein Byte im gesamten Speicherbereich angesprochen.
Um eine Doppelbelegung zu verhindern, werden Speichersysteme bzw. Ein-/Ausgabekanäle über Adreßdekoder angesprochen. Diese sind jedoch nicht Gegenstand dieses Buches.
Die bei bestimmten Mikroprozessoren möglichen Adressierungsarten sind den jeweiligen Bausteinbeschreibungen zu entnehmen.
Nachfolgende allgemeine Erläuterungen sollen einen Überblick über die möglichen Arten der Adressierung geben.

1.4.1 Unmittelbare Adressierung (immediate addressing)

Ein »immediate«-Befehl besteht bei den meisten 8-Bit-Mikroprozessoren aus zwei Byte. Hierbei folgt dem Befehlskode unmittelbar der Operand. Zwei Beispiele sollen das verdeutlichen.

Maschinenkode	Befehl		Beschreibung
A101	LDR1, 01	;	lade Register R1 mit dem Wert 01_{16}
84F0	ANDA, F0	;	führe eine UND-Verknüpfung von
		;	Akkumulator A und dem Wert $F0_{16}$
		;	durch, das Ergebnis lege in Akku A ab

Immediate Adressierung ist immer denkbar bzw. möglich, wenn man einen Akkumulator oder ein Register mit einem bei der Programmerstellung schon bekannten Operanden verknüpfen will, bzw. mit einem bekannten Wert laden will.

1.4.2 Direkte Adressierung (direct addressing)

Die Adresse des Operanden wird direkt angegeben. Sie steht hinter dem Befehl. Bei einigen Mikroprozessoren kann bei Lade- und Speicher-Befehlen nur die sogenannte »Page Zero« (die ersten 256 Speicherplätze) direkt adressiert werden. Da es sich in diesem speziellen Fall um einen Zwei-Byte-Befehl handelt, ist er meist sehr schnell.

Auch hier wollen wir uns zwei Beispiele ansehen.

JSR 8021 bedeutet: führe das Unterprogramm, welches an der Adresse 8021_{16} beginnt aus.

STAB F01A bedeutet: schreibe den Inhalt von Akkumulator B in die Speicherzelle mit der Adresse $F01A_{16}$.

Beim Mikroprozessor M6800 von Motorola wird die direkte Adressierung einer Speicherstelle außerhalb der »Page Zero« als sogenannte erweiterte (extended) Adressierung bezeichnet.

1.4.3 Relative Adressierung (relative addressing)

Die relative Adressierung wird meist bei Sprungbefehlen verwendet. Sie wird jedoch auch bei anderen Befehlen eingesetzt. Die Instruktionen sind meist Zwei-Byte-Befehle, in denen das erste Byte den Operationskode und das zweite Byte einen Zahlenwert zwischen -128_{10} und $+127_{10}$ darstellt. Dieses zweite Byte ist die relative Adresse eines Operanden oder des Sprungziels, bezogen auf den Wert des Programmzählers. Inhalt des Programmzählers plus Wert des zweiten Bytes ergibt die effektive Adresse. Im nachfolgenden Beispiel signalisiert der Punkt vor dem Displacement, daß es sich um relative Adressierung handelt. Ein solches Kürzel wird als Assemblerdirektive bezeichnet, da sie dem Übersetzungsprogramm sagt, hier muß der Kode für relative Adressierung benutzt werden.

Ein Beispiel soll uns die Zusammenhänge klar machen.

Adresse	Befehl		Beschreibung
0000	AND, .06	;	mache eine UND-Verknüpfung zwischen
0001		;	Akkumulatorinhalt und dem Inhalt der
0002	JNZ, .1A	;	Speicherstelle mit der Adresse
0003		;	Programmzähler + 06. Das Ergebnis wird
0004	:	;	im Akkumulator abgelegt. Ist der Akku-
0005	:	;	Inhalt nicht 0, so springe vom aktuellen
⋮		;	Stand des Programmzählers aus gesehen
001C	▼	;	um $1A_{16} = 26_{10}$ Adressen nach vorne.

Nehmen wir einmal an, der Programmzähler (PC) habe den Wert 1020_{16}. Mit dem Befehl »JMP .xx« kann dann jede Adresse zwischen $FA0_{16}$ und $109F_{16}$ angesprungen werden. Das Displacement »xx« stellt eine 8-Bit Zweierkomplementzahl dar.

Als kleine Hilfe kann man sich merken:
Ist die erste Ziffer des Displacements kleiner oder gleich 7_{16}, so ist es positiv.

1.4.4 Indizierte Adressierung (indexed addressing)

Die effektive Adresse ergibt sich hierbei aus dem Inhalt eines Indexregisters oder Pointerregisters (16 Bit breit) und einem Offset, der als zweites Byte des Befehls geschrieben wird. Einige Mikroprozessoren lassen bei dieser Adressierungsart nur positive Displacements zu. Es können somit mit einem Zwei-Byte-Befehl 256_{10} Speicherplätze erreicht werden. Die Anfangsadresse dieses Bereichs ist durch den Inhalt eines Indexregisters festgelegt. Dazu wieder ein Beispiel. Das Indexregister habe den Inhalt 2000_{16}.

LDAA, FO, I bedeutet: lade Akkumulator A mit dem Inhalt der Speicherzelle, deren effektive Adresse sich aus dem Inhalt des Indexregisters plus dem Displacement FO_{16} ergibt.

Im obigen Fall wird der Akkumulator A mit dem Inhalt der Speicherstelle mit der Adresse $20FO_{16}$ geladen.

1.4.5 Beinhaltende Adressierung (inherent addressing)

Bei dieser Adressierungsart, deren deutsche Übersetzung etwas unglücklich klingt, beinhaltet der Befehl selbst schon die Adresse des oder der Operanden. Es handelt sich hierbei nicht um eine echte speicherplatzbezogene Adressierung.
Drei einfache Beispiele sollen uns genügen.

CSA bedeutet:
kopiere den Inhalt des Statusregisters in den Akkumulator.

INX bedeutet:
inkrementiere den Inhalt des Indexregisters um eins.

CCL bedeutet:
setze das Carry/Link-Bit im Statusregister auf den Wert 0.

Die oben behandelten fünf unterschiedlichen Adressierungsarten werden in der beschriebenen oder einer ähnlichen Weise bei allen Mikroprozessoren angewendet. Einige Prozessoren benutzen zusätzlich noch andere Arten oder Kombinationen aus den fünf beschriebenen Versionen. Wir wollen damit das Kapitel der mehr oder weniger allgemeinen Grundlagen verlassen und uns dem Hauptthema des Buches zuwenden, dem Mikroprozessor INS8060 (SCMP) von National Semiconductor.

2. Der SCMP II (ISP-8A/600) Mikroprozessor

Nachdem über Halbleitertechnologien, grundsätzliche Zusammenhänge, Zahlendarstellungen und Adressierung gesprochen wurde, wollen wir uns jetzt mit dem »Hauptdarsteller« dieses Buches beschäftigen. Der Mikroprozessor mit der Typenbezeichnung INS8060 von National Semiconductor ist besser unter dem Namen SCMP II bekannt. Der Name SCMP steht für »**S**imple **C**ost Effective **M**icro**P**rocessor«, römisch-zwei steht für zweite Version. Im Gegensatz zur ersten Ausführung, welche in PMOS hergestellt war, ist der neue Baustein in NMOS-Technologie aufgebaut. Er ist somit doppelt so schnell als sein Vorgänger. Ein weiterer Vorteil des SCMP II ist seine einfache (+ 5V) Spannungsversorgung. Die alte CPU benötigte noch zusätzlich eine Spannung von −7V.

Der SCMP ist sicher nicht der schnellste und komfortabelste Mikroprozessor, der zur Zeit auf dem Markt ist; aber er ist, gerade für den Anfänger, recht leicht zu verstehen. Wie wir noch sehen werden, lassen sich damit doch recht leistungsfähige Programme realisieren. Für einige Anwendungen ist er auf Grund seiner einfachen Struktur sogar besser geeignet als mancher leistungsstärkere Mikroprozessor.

Den »besten« Mikroprozessor gibt es im Allgemeinen nicht, wohl aber gibt es einen für ein bestimmtes Anwendungsgebiet optimalen Mikroprozessor. Die Entscheidung, welcher Mikroprozessor für welchen Anwendungsfall, fällt schwer. Man benötigt dazu eine gute Kenntnis verschiedener auf dem Markt befindlicher Mikroprozessoren. In der Praxis kommt man um Kompromisse nicht herum, denn außer der Leistungsfähigkeit des Prozessors muß auch die Unterstützung durch Entwicklungshilfen (Entwicklungssysteme, Software u.a.) beachtet werden.

Bevor man sich als Branchenneuling mit einem Mikroprozessor auseinandersetzt, der eine große Befehlsvielfalt und eine komplizierte Struktur hat, sollte man das Prinzip der Mikroprozessorprogrammierung verstanden haben. Das wiederum geht am effektivsten mit Hilfe eines einfachen Prozessors; dem SCMP II.

2.1 Anschlußbelegung und Daten des SCMP II

Der SCMP II wird im 40poligen DIL-Gehäuse geliefert. Er ist im Plastik- oder im Keramikgehäuse erhältlich.

Die wichtigsten Informationen in Stichworten:

bidirektionaler »Tri-State« 8-Bit Datenbus,

getrennter serieller Ein- und Ausgang,

drei vom Anwender benutzbare »Steuerflag«-Ausgänge,

zwei »Fühlereingänge«, davon einer als Interrupteingang nutzbar,

der Baustein enthält Bus-Steuerlogik für Multiprozessoranwendungen und DMA,

Verzögerungsbefehl,

auf dem Chip ist eine »handshake« Bus-Zugriffssteuerung integriert, fünf Adressierungsarten,

12-Bit Adreßbus direkt verfügbar; Adreßbits 12–15 gemultiplext auf dem Datenbus,

Takterzeugung auf dem Chip,

Taktfrequenz bis 4 MHz,

alle Ein- und Ausgänge sind TTL-kompatibel,

nur eine Betriebsspannung von + 5V,

geringe Verlustleistung von nur etwa 200 mW.

Elektrische Daten:

Betriebsspannung $V_{CC} = + 5V \pm 5\%$

logisch »1« Pegel + 2V bis + 5V

logisch »0« Pegel − 0,5V bis + 0,8V

Stromverbrauch $I_{CC} = 45mA$ (alle Ausgänge unbelastet, $T_A = 25\,°C$)

Logikpegel für Adressen, Daten, NWDS und NRDS:
logisch »1« Ausgang min. + 2,4V
logisch »0« Ausgang max. + 0,4V

Logikpegel für NADS, FLAG 0–2, SOUT und NENOUT:
logisch »1« Ausgang min. + 1,5V (bei $I_{out} = - 1mA$)
logisch »0« Ausgang max. + 0,4V (bei $I_{out} = 2mA$)

Taktausgang XOUT:
logisch »1« Ausgang min. + 2,4V
logisch »0« Ausgang max. + 0,4V

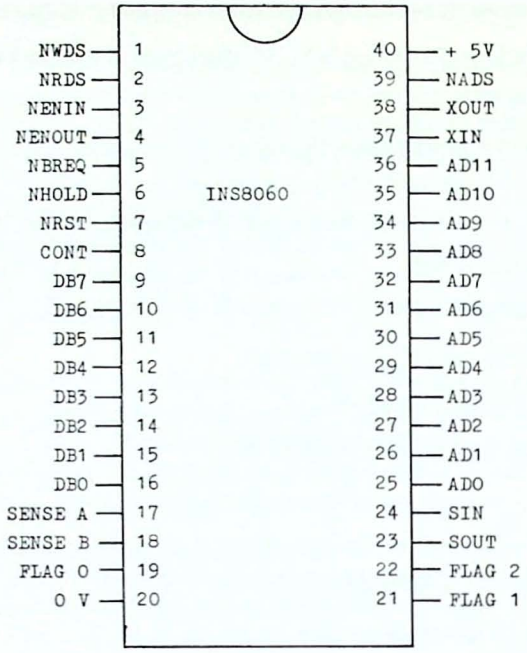


Bild 2.1.1 Anschlußbelegung des INS8060 (SCMP II)

Die in Bild 2.1.1 verwendeten Anschlußbezeichnungen werden nachfolgend ausführlich beschrieben. Die Reihenfolge ist so gewählt, daß Abhängigkeiten einzelner Signale berücksichtigt werden.

NRST (negative reset input):
Im Normalbetrieb liegt dieser Eingang auf logisch »1«. Wird er auf »0« gesetzt, so wird die gerade intern ablaufende Funktion abgebrochen. Beim Wechsel von »0« auf »1« werden alle vom Programmierer ansprechbaren Register auf Null gesetzt. Danach wird die nächste Instruktion von Adresse 0001_{16} (!) geholt. Man beachte, daß der nächste Befehl nicht von Adresse 0000_{16} geholt wird.

CONT (continue input):
Befindet sich dieser Eingang im »1« Zustand, dann arbeitet der SCMP ganz normal. Setzt man ihn auf »0«, so wird, nach Abarbeitung der gerade aktuellen Instruktion, der Prozessor angehalten. Hierbei geht der interne Status nicht verloren. Mit Hilfe dieses Signals läßt sich die normale Arbeit des Bausteins auf einfache Weise unterbrechen.

Mit den Signalen bzw. Anschlüssen NBREQ, NENIN und NENOUT läßt sich die Systembusverwaltung bei Multiprozessorsystemen und im DMA-Betrieb handhaben.

NBREQ (negative bus request input/output):
Dieser Anschluß ist mit der internen Zuteilungslogik für den Systembus verbunden. Pin 5 muß über einen Widerstand (ca. 4,7–10 kOhm) auf + 5V gelegt werden. NBREQ kann als Busanforderungsausgang oder als »Bus belegt« Eingang interpretiert werden. Während jedes Input/Output-Zyklus ist dieses Signal logisch »0«. Zusammen mit den Signalen (Anschlüsse) NENIN und NENOUT kann der Anwender beliebige Buszugriffsprioritäten festlegen. Wird der Anschluß NBREQ von außen auf »0« gelegt, so wartet der Prozessor, wenn er auf den Bus zugreifen will, bis dieser Eingang wieder logisch »1« wird. Erst danach setzt er selbst diesen Anschluß auf »0« und fährt in der Bearbeitung des gerade aktuellen Befehls fort.

NENIN (negative enable input):
Auch dieser Anschluß ist mit der internen Buszuteilungslogik verbunden. Wird der Eingang auf »0« gelegt, dann ist dem SCMP der Zugriff zum Systembus erlaubt. Ist NENIN logisch »1«, so bedeutet das für den Prozessor, der Systembus ist im »Tri-State«-Mode.

NENOUT (negative enable output):
Der dritte Anschluß, der an die interne Buszuteilungslogik angeschlossen ist, steht an Pin 4 zur Verfügung. Zusammen mit den Signalen NENIN und NBREQ läßt sich das Signal NENOUT zur Bussteuerung in Multiprozessorsystemen und für den DMA-Betrieb benutzen.
NENOUT wird »0«, wenn NENIN gleich »0« ist und der SCMP nicht den Systembus benutzt (NBREQ gleich »1«). Zu allen anderen Zeiten ist das Signal »1«.

Für die Daten Input/Output-Steuerung (für den Prozessor sind auf dieser Ebene auch Adressen Daten) gibt es vier Ein-/Ausgänge.

NADS (negative address strobe output):
Dieser Ausgang signalisiert, wenn er »0« ist, daß eine gültige Adresse und der Prozessorstatus am Systembus anliegt. Die Signaldauer beträgt bei 2 MHz minimal 200 ns.

NRDS (negative read data strobe output):
Ist kein Input/Output-Zyklus aktiv, so nimmt dieser Ausgang den »Tri-State«-Zustand ein. Bei einem Lesezyklus wird NRDS intern für eine Zeit von min. 550 ns (bei 2 MHz) auf »0« geschaltet. Mit der »0/1« Flanke des Strobes werden die Daten vom Datenbus in den Prozessor übernommen. Sobald bei einem Lesezyklus der SCMP eine gültige Adresse auf den Adreßbus gelegt hat, geht NRDS vom hochohmigen Zustand in den logischen »1« Zustand.

NWDS (negative write data strobe output):
Auch dieser Signalausgang ist mit einem »0« Signal aktiv. Er ist ebenso wie NRDS zwischen den Input/Output-Zyklen im »Tri-State« Zustand. Für eine Zeit von minimal 500 ns (bei 2 MHz) wird NWDS gleich »0« und zeigt damit an, daß gültige Daten auf dem Datenbus liegen.

NHOLD (negative hold input or negative input/output cycle extend input):
Über diesen Eingang kann der Schreib- oder Lesezyklus so verlängert werden, daß auch langsame Speicher oder periphere Geräte direkt mit der CPU verkehren können. Wird NHOLD vor der »0/1« Flanke des NRDS- bzw. NWDS-Strobes auf »0« gesetzt, dann bleibt der Strobe solange im aktiven Zustand wie das NHOLD-Signal auf »0« liegt.

Nach den Steuersignalen wollen wir uns mit den restlichen Anschlüssen befassen.

AD0 - AD11 (address bit 0 through 11):
Ist das NADS-Signal aktiv, so werden vom SCMP die niederwertigsten 12 Adreßbits auf den Adreßbus geschaltet. Die Adreßinformation bleibt bis zur »0/1« Flanke des NRDS- oder NWDS-Strobes anstehen. Danach nehmen diese 12 Ausgänge den »Tri-State« Zustand bis zum nächsten NADS-Signal ein. Der SCMP kann demnach mit diesen 12 Adreßbits 4KByte (4096_{10} Speicherstellen) direkt ansprechen.

Der Datenbus wird nicht nur für den Datentransport verwendet. Er enthält zu bestimmten Zeiten auch Adressen und Statusinformationen. Aus diesem Grund wollen wir uns jedes einzelne Bit des Datenbusses getrennt anschauen.

DB0–DB7 (data bit 0 through 7):
Während des NRDS-Strobes werden vom SCMP an den acht Anschlüssen Daten erwartet. Zur Zeit des NWDS-Strobes stehen gültige Daten an den Pins DB0–DB7 zu Verfügung.

Solange das Strobesignal NADS ansteht, haben die Datenanschlüsse die nachfolgende Bedeutung.

DB0 liefert das Adreßbit AD12
DB1 liefert das Adreßbit AD13
DB2 liefert das Adreßbit AD14
DB3 liefert das Adreßbit AD15
DB4 zeigt den Status des R-Flags an. Dieses ist »1«, wenn vom Prozessor ein Lesezyklus (read cycle) gestartet wurde. Beim Schreibzyklus (write cycle) ist das Flag »0«.
DB 5 zeigt den Status des I-Flags an. Es ist nur »1«, wenn das erste Byte einer Instruktion aus dem Speicher geholt wird.

DB6 zeigt den Status des D-Flags. Hat das Flag den Zustand »1«, so wird damit signalisiert, daß gerade der Verzögerungsbefehl (DLY) ausgeführt wird. Es wird »1«, wenn das zweite Byte der DLY-Instruktion aus dem Speicher geholt wird.

DB7 zeigt den Status des H-Flags. Ein »1« Zustand zeigt an, daß gerade die HALT-Instruktion ausgeführt wurde. Wird der Zustand dieses Flags in einem »Latch« zwischengespeichert, so kann zusammen mit dem CONT-Eingang ein programmierbarer HALT realisiert werden.

Es folgen die Anschlüsse des SCMP die per Programm angesprochen werden können. Insgesamt sind es sieben Signale bzw. Anschlüsse.

FLAG 0, 1, 2 (flag outputs):
Hierbei handelt es sich um vom Anwender frei benutzbare Ausgabesignale des Statusregisters. Die Flags können gelöscht bzw. gesetzt werden, indem per Instruktion der Akkumulatorinhalt ins Statusregister kopiert wird. Die Ausgänge können jeweils eine TTL-Last treiben.

SIN (serial input):
Daten an diesem Anschluß werden mit dem Befehl SIO in das höchstwertige Bit des Extensionregisters geschoben.

SOUT (serial output):
Das niederwertigste Bit des Extensionregisters wird mit dem Befehl SIO an den Ausgang SOUT geschoben. Der Zustand dieses Bits wird über ein Flipflop (Latch) bis zum nächsten SIO gespeichert.

SENSE A (sense input):
Solange das IE (Interrupt Enable) Bit im Statusregister »0« ist, wird dieser Eingang als normaler Fühlereingang behandelt. Der Zustand kann abgefragt werden, in dem man das Statusregister in den Akku kopiert.
Ist das IE-Bit »1«, so wirkt SENSE A als Interrupteingang. Eine »1« an diesem Anschluß bewirkt dann die Ausführung des Befehls XPPC3 (tausche den Inhalt des Programmzählers mit dem Inhalt von Pointer 3). Steht in Pointer 3 die Adresse eines Programms, so wird dieses dadurch aufgerufen und ausgeführt.

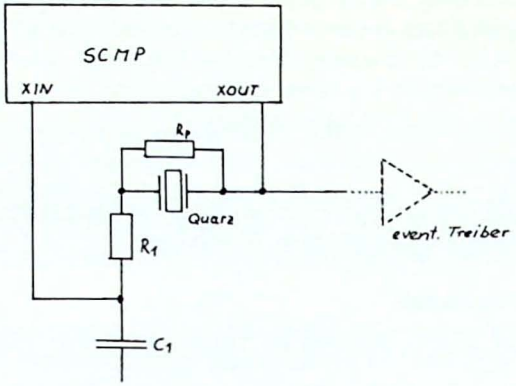
SENSE B (sense input):
Fühlereingang der durch Kopieren des Statusregisters in den Akkumulator abgefragt werden kann.

SENSE A und B sind reine Eingänge, die nicht per Programm gesetzt werden können.

Die letzten noch nicht besprochenen Anschlüsse des SCMP sind die Pins für die Betriebsspannung + 5V und OV an Pin V_{CC} bzw. GND. Außerdem noch XIN und XOUT

für den Takt. Bild 2.1.2 zeigt eine typische Beschaltung dieser Anschlüsse für Anwendungen mit Frequenzen über 1 MHz.

Bild 2.1.2:



Die Zusammenhänge zwischen Quarzfrequenz und Widerständen bzw. Kondensator sind in nachfolgender Tabelle aufgelistet. Der Quarz ist ein Typ mit Parallelresonanzcharakteristik.

Quarz	R_p	C_1	R_1
2 MHz	100 kOhm	56 pF	1 kOhm
3,58 Mhz	100 kOhm	27 pF	1 kOhm
4 MHz	100 kOhm	27 pF	1 kOhm

2.2 SCMP-Architektur

Gegenüber anderen 8-Bit Mikroprozessoren hat der SCMP eine relativ einfache Struktur. In Bild 2.2.1 ist der innere Aufbau des Prozessors als Blockdiagramm dargestellt. Da die englische Bezeichnung in der Mikroprozessortechnik (Datentechnik allgemein) üblich ist, werden auch hier diese Bezeichnungen verwendet. Der SCMP enthält zehn interne Register. Auf sieben dieser Register kann der Benutzer zugreifen. Sie werden in Kapitel 2.2.1 bis 2.2.4 beschrieben.
Befassen wir uns darum mit den anderen.
Das INSTRUCTION REGISTER wird während jeder »Hol-Phase« eines Befehlszyklus mit dem acht Bit breiten Operationskode aus dem Speicher geladen. Dieser Kode steht dann zur internen Weiterverarbeitung durch den Befehlsdekoder zur Verfügung.
Zum temporären Zwischenspeichern aller Ein-/Ausgabedaten, die über den acht Bit breiten Datenbus kommen oder gehen dient das DATA I/O REGISTER. Dies geschieht

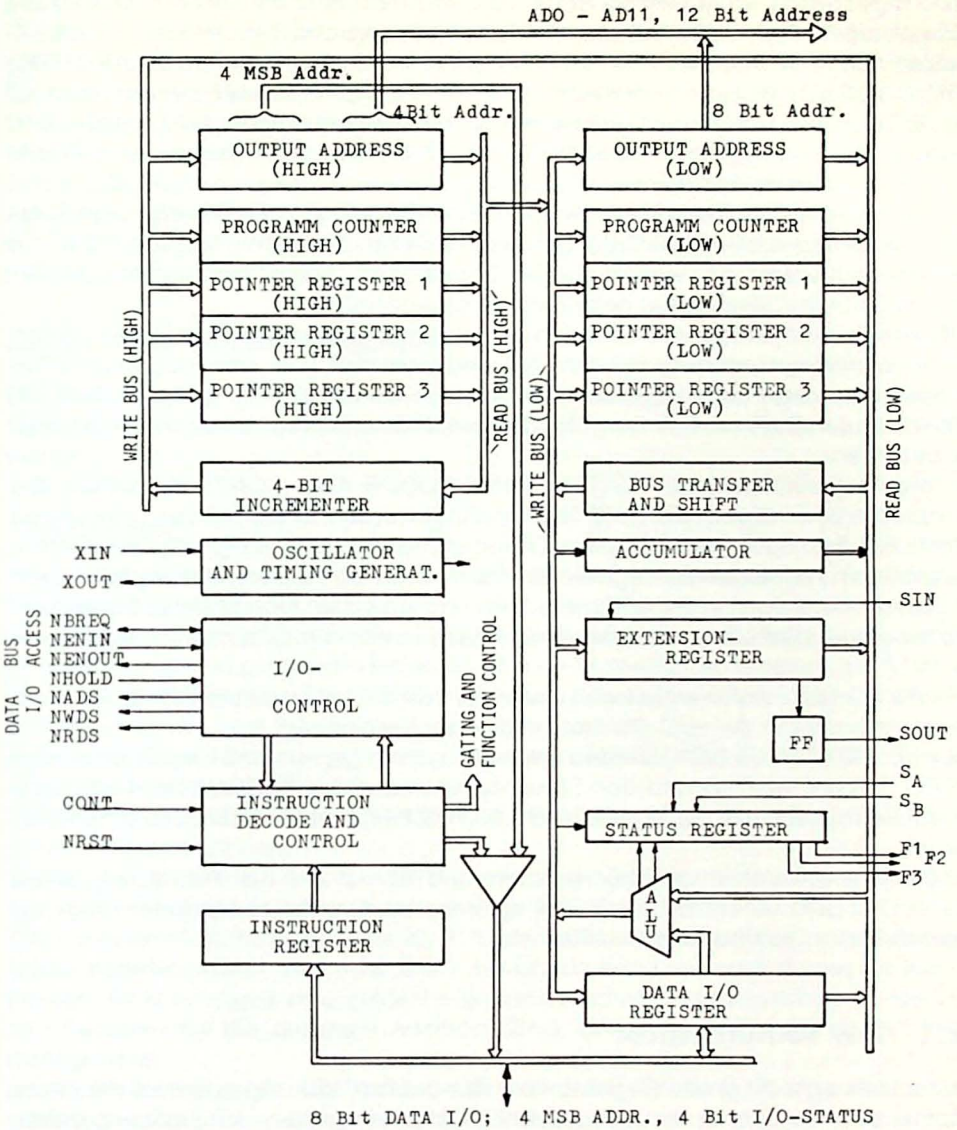


Bild 2.2.1: Blockdiagramm des INS8060 (SCMP II) Mikroprozessors

während der »Daten-Transfer-Phase« eines jeden Ein-/Ausgabezyklus (NRDS oder NWDS logisch »0«). Im ADDRESS REGISTER, welches sechzehn Bit breit ist, wird die im Ein-/Ausgabezyklus übertragene 16-Bit Adresse zwischengespeichert.
Eine zentrale Rolle innerhalb der CPU (central processing unit) spielt die ARITHMETIC AND LOGIC UNIT auch bekannt unter der Abkürzung ALU. Die ALU liefert die Möglichkeit, Daten zu verarbeiten. Operationen, die durch die ALU ermöglicht werden, sind alle logischen Verknüpfungen (OR, XOR, AND), Inkrement- und Dekrement-Befehle, binäre und dezimale Addition.
Über und mit der BUS TRANSFER AND SHIFT LOGIC werden alle Schiebe- und Rotationsbefehle und das serielle Datenhandling realisiert. Außerdem werden damit die Busaustauschfunktionen, wie sie für die Datenbewegungen zwischen den SCMP-internen Schreib-/Lesebussen notwendig sind, gesteuert.
Zu diesen Funktionen gehören außer den bereits genannten die Austauschbefehle, die die Pointerregisterinhalte mit dem Akkumulatorinhalt, bzw. den Programmzähler mit einem der drei Pointerregisterinhalte austauschen. Weiterhin gehören dazu die Befehle, die den Datenverkehr zwischen Akkumulator und Extension- oder Statusregister betreffen.
Der interne Funktionsblock INSTRUCTION DECODE AND CONTROL enthält das Mikroprogramm, mit dem alle Befehle ausgeführt werden. Er steuert das Lesen eines Befehls aus dem Speicher und dessen Dekodierung. Weiterhin sorgt er für den Datentransport z.B. vom Extensionregister über das »Data I/O Register« in die ALU. Nach der Datenverknüpfung in der arithmetischen und logischen Einheit (ALU) steuert der Funktionsblock den Abtransport des Ergebnisses an das durch den Befehl festgelegte Ziel.
Die 4-Bit I/O-Statusinformation wird ebenfalls von dieser Einheit geliefert. Sie steht außerdem noch mit der »I/O Control« in direkter Verbindung.
In der I/O CONTROL LOGIC werden schließlich abhängig von den Eingangssignalen NBREQ, NENIN, NHOLD und den Steuersignalen aus der INSTRUCTION DECODE AND CONTROL Einheit, die Signale NADS, NWDS, NRDS sowie NBREQ und NENOUT erzeugt.
Detailliertere Informationen über den internen Ablauf bei der Ausführung eines Befehls sind dem Datenblatt des SCMP zu entnehmen, welches von jedem National Semiconductor Distributor zu erhalten ist.

2.2.1 Der Akkumulator

Das zentrale acht Bit breite Register, über das bei fast allen Operationen die Daten ausgetauscht werden, ist der Akkumulator. Er wird im Weiteren mit »Akku« bezeichnet.
Der Name Akkumulator kommt aus dem Lateinischen und heißt soviel wie Sammler. Wir können festhalten, der Akku ist das in erster Linie benutzte Arbeitsregister des SCMP.

2.2.2 Das Extension-Register

Dieses Register ist, wie die englische Bezeichnung schon sagt, ein Erweiterungsregister. Es stellt eine Ergänzung des Akkus dar und ist ebenso wie dieser acht Bit breit. Das Extensionregister wird bei der seriellen Datenein- und -ausgabe benutzt. Das höchstwertige und das niederwertigste Bit dieses Registers sind an Anschlußpins des SCMP herausgeführt (SIN und SOUT). SOUT wird über ein Flipflop intern gepuffert. Noch eine Besonderheit des Extensionregisters. Ist bei indizierter bzw. autoindizierter Adressierung das zweite Byte des Befehls (es gibt das Displacement an) gleich 80_{16} ($= -128_{10}$), so wird der Inhalt des Extensionregisters als Displacement zur Bestimmung der effektiven Adresse benutzt.

2.2.3 Das Status-Register

Das Statusregister bietet Platz für acht verschiedene Flags, die durch arithmetische Befehle, durch gezieltes Ansprechen per Software oder von der Peripherie beeinflußt werden.

	Bit 7	6	5	4	3	2	1	0
Statusregister	CY/L	OV	S_B	S_A	IE	F2	F1	F0

Bit 0, 1 und 2 enthalten die vom Anwender über Befehl beliebig setzbaren Ausgabeflags F0, F1 und F2.
Bit 3 ist das »Interrupt Enable Flag« (IE). Auch dieses interne Statusbit kann per Programm gesetzt und gelöscht werden. Ist es auf »1« gesetzt, so reagiert der SCMP auf Interruptsignale, die über den SENSE A Eingang angelegt werden, indem er den Programmzähler mit dem Inhalt von Pointer 3 tauscht.
Setzt man IE auf »0«, so sind Interrupts gesperrt, SENSE A ist ein normaler Eingang. Doch dazu mehr in Kapitel 2.3.
Die Bits 4 und 5, SENSE A und SENSE B, sind reine Eingänge, die nicht per Programm gesetzt werden können.
SENSE A übernimmt, wenn das IE-Flag »1« ist, die Rolle eines Interrupteingangs.
Wenden wir uns Bit 6, dem »Overflow-Flag« (OV) zu.
Tritt bei einem Additionsbefehl (ADD, ADI oder ADE) oder einem Komplementier- und Additionsbefehl (CAD, CAI oder CAE) ein Überlauf auf, so wird dieses Bit auf »1« gesetzt. Es kann damit eine Zahlenbereichsüberschreitung festgestellt werden. Bei den Befehlen für die dezimale Addition (DAD, DAI oder DAE) wird das »OV« Bit nicht gesetzt.
Das letzte Statusbit, also Bit 7, ist das »Carry/Link Bit« (CY/L). Es kann gezielt per Programm gesetzt (SCL) oder gelöscht (CCL) werden.
Immer, wenn bei Additionsbefehlen bei der stellenweisen Addition ein Übertrag im höchstwertigen Bit bzw. von der 16^1- zur 16^2-Stelle auftritt, wird das »CY/L« Bit auf »1« gesetzt. Tritt kein Übertrag auf, so erhält dieses Bit den Wert »0«.
Die Bedeutung dieses Bits in der Software wird in Kapitel 3 ausführlich besprochen.

2.2.4 Der Programm Counter (PC)

Das Wort Programmschrittzähler, was die sinngemäße deutsche Übersetzung des englischen Begriffs ist, deutet schon auf die Aufgabe dieses Registers hin. Die allgemein übliche Abkürzung für Programm Counter ist »PC«.
Der PC ist ein 16 Bit breites Register, welches die Adresse des gerade ausgeführten Befehls enthält. Jedesmal bevor ein Befehl aus dem Speicher geholt wird, wird der Inhalt dieses Registers automatisch um eins erhöht. Dadurch ist der sequentielle Ablauf eines Programms gewährleistet. Durch Instruktionen, die wir noch kennenlernen werden, kann der PC gezielt verändert werden. Es sind damit Unterprogrammaufrufe und Programmverzweigungen möglich.
Eine Besonderheit beim Programm Counter des SCMP ist, daß das automatische Inkrementieren des PC-Inhaltes nur bei den unteren 12 Bit stattfindet. Dadurch kann der SCMP nur einen Adreßbereich von 4 KByte (4096_{10}) fortlaufend abarbeiten. Ein Übertrag in die vier höchstwertigen Adreßbits erfolgt nicht. Will man einen solchen 4 KByte Bereich, auch »Seite« oder »Page« genannt überschreiten, so muß man an der Stelle im Programm an der der Seitenwechsel stattfinden soll, den Programm Counter gezielt setzen. Obwohl diese Tatsache vielleicht zunächst als Nachteil angesehen wird, so bietet sie doch eine gewisse Sicherheit gegen sogenannte »Killerprogramme«. Diese können unter Umständen den gesamten Speicher durchlaufen und dabei andere Programme zerstören. Die vorher genannten vier höchstwertigen Bits bilden somit einen Seitenzähler, der insgesamt 16 Seiten je 4 KByte selektieren kann. Damit ist eine Adressierung über einen Bereich von 64 KByte (65536_{10}) möglich.

2.2.5 Die Pointer-Register P1, P2 und P3

Auch diese drei Register sind 16-Bit Register. Sie werden normalerweise im Programm mit bestimmten Referenzadressen geladen, die dann als »Stackpointer« (Stapelzeiger), »Page Pointer« (Seitenzeiger) oder Unterprogrammzeiger dienen. Selbstverständlich können sie auch als Datenregister verwendet werden.
Das Pointerregister 3 hat zusätzlich noch eine weitere Aufgabe. Bei einem Interrupt an SENSE A wird vom Mikroprogramm der Programm Counter auf den Inhalt des Pointerregisters 3 gesetzt und damit diese Adresse angesprungen.

2.3 Interruptbehandlung beim SCMP

Solange das Interrupt Enable Bit »IE« im Statusregister den Wert »0« hat, sind keine Interrupts erlaubt. Die beiden »Fühlereingänge« SENSE A und SENSE B werden als einfache Eingänge betrieben. Mit dem Befehl »DINT« (disable interrupts) kann dieser Zustand herbeigeführt werden. Setzt man nun das Bit »IE« mit dem Befehl »IEN« (interrupt enable) auf »1«, so werden Unterbrechungen zugelassen. Der SENSE A Eingang

wirkt jetzt als Interrupteingang. Eine weitere Möglichkeit, den Wert des Interrupt Enable Bits im Statusregister zu beeinflussen, besteht darin, daß man den Akku mit einem entsprechenden Bitmuster lädt und ihn dann in das Statusregister kopiert (CAS-Befehl). Bei dieser Vorgehensweise muß dafür gesorgt werden, daß sich der Zustand der anderen Statusbits nicht unerwünscht ändert. Eine solche Programmierung hat außerdem noch den Nachteil, daß ein Programm leicht unübersichtlich wird. Doch zurück zur Interruptbehandlung.
Ist eine Unterbrechung erlaubt, so wird vom Prozessor der Pointer 3 als Zeiger zu einer »Interrupt Service Routine« (Programm welches die Unterbrechungsanforderung bearbeitet) benutzt. Sobald der SENSE A Eingang den Wert »1« annimmt, wird vom Mikroprogramm der Befehl »XPPC3« ausgeführt. D.h., der Inhalt vom Programm Counter wird mit dem Inhalt des Pointers 3 ausgetauscht. Da nach diesem Befehl der neue PC automatisch gleich um eins erhöht wird, muß der Pointer 3 die um eins verringerte Adresse der Interrupt Service Routine enthalten, bzw. der erste Befehl in dieser Routine muß eine »Leerinstruktion« (NOP) sein. Nach der Reaktion auf die Unterbrechungsanforderung (XPPC3) sperrt der Prozessor neue Interrupts, indem er das »IE«-Bit auf »0« setzt.
Vor dem Rücksprung (erneuter Befehl XPPC3) aus dem aufgerufenen Programm sollten neue Interrupts durch die Instruktion »IEN« wieder freigegeben werden.
Führt man mehrere externe Unterbrechungsleitungen über einen Prioritätsenkoder auf den SENSE A Eingang, so kann man in der Interrupt Service Routine zunächst abfragen, von wem die Anforderung kommt, um dann in das entsprechende Programm zu verzweigen. Trotz der relativ einfachen Interruptstruktur des SCMP kann man auf diese Weise recht komplexe Interruptsteuerungen aufbauen.

3. Die Befehle des SCMP

Der SCMP hat insgesamt 46 unterschiedliche Basisbefehle. Nimmt man jedoch alle möglichen Kombinationen dazu, so kommt man auf 122 verschiedene Befehlskodes. Einundzwanzig der Grundbefehle sind »Ein-Byte-Befehle«, die restlichen fünfundzwanzig Instruktionen sind »Zwei-Byte-Befehle«. Die Befehlsausführungszeiten, bei einer Taktfrequenz von 2 MHz, reichen von 10 Mikrosekunden für den Befehl »NOP« bis zu 46 Mikrosekunden für die Instruktion »DAD«. Eine Besonderheit bildet der Verzögerungsbefehl »DLY«. Er hat eine Ausführungszeit von 26 Mikrosekunden bis 263,186 Millisekunden.

3.1 Die Adressierungsarten des SCMP

Unser Mikroprozessor hat fünf unterschiedliche Adressierungsmöglichkeiten. In der Mikroprozessortechnik ist es, wie schon gesagt, üblich englische Bezeichnungen zu wählen. Wir wollen uns auch daran halten.

3.1.1 Implied (beinhaltende) Adressierung

Diese wird auch als Inherent Adressierung bezeichnet. Es handelt sich dabei um keine »echte« Adressierung, bei der eine Speicherstelle angesprochen wird. Vielmehr wird in diesem Fall der Akku, das Extensionregister, das Statusregister und/oder die Pointerregister adressiert. Der Befehl selbst beinhaltet die Adresse(n). Instruktionen, die die »Implied Adressierung« benutzen, sind nachfolgend in alphabetischer Reihenfolge aufgelistet.

ADE; ANE; CAE; CAS; CCL; CSA; DAE; DINT; HALT; IEN; LDE; ORE; RR; RRL; SCL; SIO; SR; SRL; XAE; XRE; XPAH; XPAL; XPPC

3.1.2 Immediate (unmittelbare) Adressierung

Auch bei dieser Adressierungsart wird kein Speicherplatz adressiert. Alle »Immediate-Befehle« beziehen sich auf den Akkumulator und sind »Zwei-Byte-Befehle«. Das zweite Byte stellt den Operanden dar. Als Beispiel sei der LDI (load immediate) Befehl genannt. LDI 00 besagt, daß der Akku mit dem Wert 00_{16} geladen werden soll. Die Immediate-Adressierung wird bei arithmetischen und logischen Befehlen und beim Verzögerungsbefehl angewendet. Es folgt eine Übersicht dieser Befehle.

ADI; ANI; CAI; DAI; DLY; LDI; ORI; XRI

3.1.3 PC-relative Adressierung

Bei dieser Adressierung gibt das zweite Byte des Befehls ein Displacement (Versatz) an, welches im Zweierkomplement dargestellt ist. Die effektive Speicheradresse ergibt sich aus der Addition von Displacement und aktuellen Programmzählerwert. Als PC-Stand gilt der an der das Displacement steht. Dazu folgendes Beispiel.

Adresse	Inhalt	Befehl	Kommentar
0100	AA		; sedezimale Daten
0101	08	NOP	; Leerbefehl
0102	CO	LD FD	; lade Inhalt von Adresse 0100_{16}
0103	FD		;
0104	C8	ST 03	; speichere Akku-Inhalt nach
0105	03		; Adresse 0108_{16}
0106	00	HALT	; HALT-Instruktion
0107	xx	·	; beliebiger Inhalt
0108	xx	·	; beliebiger Inhalt

Der Inhalt des Bytes mit der Adresse 0100_{16} (AA_{16}) wird nach der Speicherstelle mit Adresse 0108_{16} geschrieben. Das Displacement FD_{16} (-3_{10}) steht an der Adresse 0103_{16} und spricht damit die Speicherstelle 0100_{16} ($0103_{16}-03_{16} = 0100_{16}$) an. Der Inhalt des Akkumulators (er ist nach dem Ladebefehl AA_{16}) soll in die Speicherzelle mit der Adresse 0108_{16} geschrieben werden.

Zu der Zeit in der der Befehl »ST« an der Adresse 0104_{16} vom Mikroprogramm entschlüsselt wird, steht der Programmzähler schon auf dem Wert 0105_{16}. Das Displacement bezieht sich demzufolge auf diesen Wert. Daraus ergibt sich die effektive Adresse mit $0105_{16} + 03_{16} = 0108_{16}$. Nach Ausführung der »ST«-Instruktion steht in Adresse 0108_{16} der Wert AA_{16}.

Bei der PC-relativen Adressierung können Speicherstellen im Bereich von -128_{10} bis $+127_{10}$ relativ zum aktuellen Programmzählerstand angesprochen werden.

Hat das Displacement den sedezimalen Wert »80«, so wird zur Berechnung der effektiven Adresse der Inhalt des Extensionregisters benutzt. D.h., auf unser Beispiel bezogen, daß wir z.B. beim ST-Befehl das gleiche Ergebnis erzielen, wenn der Offset anstelle von 03_{16} den Wert 80_{16} hat und gleichzeitig der Inhalt des Extensionregisters 03_{16} ist.

3.1.4 Indexed (indizierte) Adressierung

Mit Hilfe der indizierten Adressierung kann der gesamte Speicherbereich von 64 KByte (0000_{16} bis $FFFF_{16}$) adressiert werden. Die Pointerregister P1, P2 und P3 dienen als Basis- bzw. Indexregister. Zur Ermittlung der effektiven Adresse wird das Displacement (zweites Byte des Befehls) zum Inhalt des entsprechenden Pointerregisters addiert. Das Ergebnis bildet die absolute Adresse. Der Inhalt des Pointerregisters wird nicht verändert. Bezüglich des Displacements gilt das Gleiche wie bei der PC-relativen Adressierung.

Zu beachten ist, daß der SCMP von sich aus keine sogenannte »Page« verlassen kann. Hat z.B. Pointer 2 den Inhalt 2000_{16} und wird der Befehl LD FF (2) ausgeführt, so ist die effektive Adresse nicht $1FFF_{16}$ sondern $2FFF_{16}$.

Eine Untergruppe der indizierten Adressierung ist die »autoindizierte Adressierung«. Es gelten die gleichen Regeln. Der Unterschied liegt darin, daß mit Ausführung eines solchen Befehls der Inhalt des entsprechenden Pointers durch das Displacement verändert wird. Ist der **Offset negativ,** so wird der Pointer **vor** Ausführung des Befehls um den Wert des Offsets **dekrementiert.** Bei **positivem** Displacement wird der Pointer erst **nach** Ausführung der Operation um den Wert des Displacements **inkrementiert.** Mit Hilfe dieser Adressierungsart lassen sich relativ einfach Tabellen verarbeiten. Auch für Stack-Operationen ist die autoindizierte Adressierung sehr nützlich.

Im Abschnitt »Programme« wird dieser Adressierungsmodus in Beispielprogrammen angewendet.

3.2 Speicherbezogene Befehle

Zunächst eine alphabetische Liste der möglichen Basisbefehle.

ADD; AND; CAD; DAD; LD; OR; ST; XOR

Nachfolgend werden die oben genannten Instruktionen ausführlich besprochen. Die Auswahl der Reihenfolge wurde nach didaktischen Gesichtspunkten ausgewählt.

Das Zeichen @ signalisiert »autoindizierte Adressierung«.

3.2.1 Der Lade-Befehl LD (load)

Bei diesem Befehl sind PC-relative, Pointer indizierte und autoindizierte Adressierung möglich. Die Variationen dieser Instruktion sind in einer kleinen Tabelle zusammengestellt.

Mnemonischer Befehl	Operationskode	Beschreibung
LD xx	C0 xx	bezogen auf Programm Counter
LD xx (P1)	C1 xx	Pointer 1 indiziert
LD xx (P2)	C2 xx	Pointer 2 indiziert
LD xx (P3)	C3 xx	Pointer 3 indiziert
LD @xx (P1)	C5 xx	Autoindiziert Pointer 1
LD @xx (P2)	C6 xx	Autoindiziert Pointer 2
LD @xx (P3)	C7 xx	Autoindiziert Pointer 3

Operation: **(EA) → (AC)**

xx steht für ein Displacement, dessen Wert zwischen 00_{16} und FF_{16} (— 128_{10} bis + 127_{10}) liegen kann.
(...) bedeutet Inhalt von...
Diese Nomenklatur wird auch in den weiteren Beispielen verwendet. Sie wird deshalb nicht jedesmal neu erklärt.

Der Akku wird mit dem Inhalt der effektiven Adresse (EA) geladen. Der Inhalt dieser Speicherstelle wird durch den Befehl nicht verändert. Für die PC-relative Adressierung steht ein Beispiel in 3.1.3. Für die Pointer indizierte und autoindizierte Adressierung folgen welche.
Dazu müssen wir Annahmen treffen.

Pointer 1 = 1000_{16}, Pointer 2 = $200A_{16}$, Pointer 3 = $3FFF_{16}$

Nach Ausführung der Instruktion 'LD 00 (P1)' steht im Akku der Inhalt der Adresse 1000_{16}. Heißt der Befehl 'LD 7F (P1)', so wird der Akku mit dem Inhalt der Speicherstelle $107F_{16}$ geladen. Noch ein Beispiel mit Pointer 3. Mit 'LD 05 (P3)' wird der Inhalt der Adresse 3004 (nicht 4004!) in den Akkumulator gebracht.

Für die autoindizierte Adressierung gilt folgendes.

'LD @01 (P1)' lädt den Inhalt der Adresse 1000_{16} in den Akku. Nach dem Laden wird Pointer 1 um eins inkrementiert. Sein neuer Wert ist 1001_{16}. D.h., nach jedem Ausführen dieses Befehls wird Pointer 1 um eins erhöht. Hat das Displacement einen anderen Wert, so wird der Pointer um diesen Wert inkrementiert.

Ist das Displacement negativ, so gilt:

Nach 'LD @FF (P2)' enthält der Akkumulator den Inhalt der Adresse 2009_{16} und Pointer 2 hat den Wert 2009_{16}. Bei negativem Offset wird vor dem Laden der Pointer um das Displacement dekrementiert.

3.2.2 Der Speicher-Befehl ST (store)

Bezüglich der Adressierung sind die gleichen Regeln wie beim »LD« gültig. Auch hier wurde der Befehl im Zusammenhang mit der PC-relativen Adressierung in Kapitel 3.1.3 besprochen.

Wie beim Lade-Befehl gibt es auch hier sieben Modifikationen der Instruktion.

Mnemonischer Befehl	Operationskode	Beschreibung
ST xx	C8 xx	bezogen auf Programm Counter
ST xx (P1)	C9 xx	Pointer 1 indiziert
ST xx (P2)	CA xx	Pointer 2 indiziert
ST xx (P3)	CB xx	Pointer 3 indiziert
ST @xx (P1)	CD xx	Autoindiziert Pointer 1
ST @xx (P2)	CE xx	Autoindiziert Pointer 2
ST @xx (P3)	CF xx	Autoindiziert Pointer 3

Operation: **(AC)** → **(EA)**

Da sich der ST-Befehl bezüglich angesprochener bzw. anzusprechender Adresse nicht vom LD-Befehl unterscheidet, soll uns ein Beispiel genügen. Es gelten die angenommenen Pointerwerte aus 3.2.1. Die Instruktion »ST @01 (P1)« bewirkt, daß der Inhalt des Akkus in die Adresse 1000_{16} geschrieben wird. Der Pointer 1 hat nach der Ausführung den Wert 1001_{16}. Der Akkumulatorinhalt bleibt erhalten. Eine Wiederholung des Befehls, z.B. in einer Schleife, bewirkt, daß beim nächsten Mal die Adresse 1001_{16}, dann 1002_{16} usw. beschrieben wird. Damit können auf elegante Weise Tabellen verarbeitet werden.

3.2.3 Die UND-Verknüpfung AND (and)

Beim »AND«-Befehl sind die gleichen Adressierungsarten wie bei den Lade- und Speicher-Befehlen möglich.

Mnemonischer Befehl	Operationskode	Beschreibung
AND xx	D0 xx	Programm Counter relativ
AND xx (P1)	D1 xx	Pointer 1 indiziert
AND xx (P2)	D2 xx	Pointer 2 indiziert
AND xx (P3)	D3 xx	Pointer 3 indiziert
AND @xx (P1)	D5 xx	Autoindiziert Pointer 1
AND @xx (P2)	D6 xx	Autoindiziert Pointer 2
AND @xx (P3)	D7 xx	Autoindiziert Pointer 3

Operation: **(AC)** $\wedge$ **(EA)** $\rightarrow$ **(AC)**

$\wedge$ Zeichen für logisches UND

Der Inhalt des Akkus wird mit dem Inhalt der effektiven Adresse UND verknüpft. Das Ergebnis dieser Operation steht im Akkumulator. Der alte Inhalt der adressierten Speicherstelle bleibt erhalten.

Annahme: Pointer 1 = $A100_{16}$, Pointer 2 = 2000_{16}
Der Inhalt der Speicherstelle mit der Adresse $A100_{16}$ sei z. B. $D2_{16}$ und der Inhalt der Adresse 2000_{16} sei $7F_{16}$.

Man benutzt in der Praxis, für die Inhaltsangabe von Speicherstellen, auch die folgende Schreibweise.

$$(A100_{16}) = D2_{16}, (2000_{16}) = 7F_{16}$$

Es handelt sich bei den Werten um willkürliche Werte.

Lassen wir den SCMP die nachfolgenden Instruktionen abarbeiten und schauen wir uns an, was dabei geschieht.

```
Programmzeile 1: LD 00 (P1)      ;  Akku erhält den Wert D2(16)
Programmzeile 2: AND 00 (P2);  Inhalt des Akkus wird mit dem Inhalt von
                                 ;  Adresse 2000(16) UND-verknüpft. Das Ergebnis
                                 ;  steht im Akku
Programmzeile 3: ST 50 (P1)      ;  Ergebnis in Adresse A150(16) abspeichern.
```

Zur Verdeutlichung schauen wir uns das Ganze einmal auf Bitebene an.

Programmzeile 1: $(A\ 100_{16}) = 11010010_2 = D2_{16} \rightarrow (AC)$
Programmzeile 2: $(AC) = (AC) \wedge (2000_{16}) = 01010010_2 = 52_{16}$
Programmzeile 3: $(A150_{16}) = (AC) = 52_{16}$

Mit dem AND-Befehl können einzelne Bits oder Bitmuster aus einem gegebenen Bitmuster ausgeblendet werden.
Bei dieser Instruktion wird Bit für Bit UND-verknüpft. Für unser Beispiel sieht das wie folgt aus.

$$\begin{array}{ll} & 11010010_2 = D2_{16} \\ \text{»AND«} & 01110010_2 = 7F_{16} \\ \hline & 01010010_2 = 52_{16} \end{array}$$

Im Zusammenhang mit den Sprungbefehlen wollen wir uns anschauen, wie man die UND-Verknüpfung zusammen mit einem nachfolgenden bedingten Sprung für eine Abfrage benutzen kann.

Anhand von Bild 3.2.3.1 soll einmal kurz verdeutlicht werden, welchen Schaltungsaufwand man treiben muß, um eine solche achtstellige UND-Verknüpfung zu realisieren.

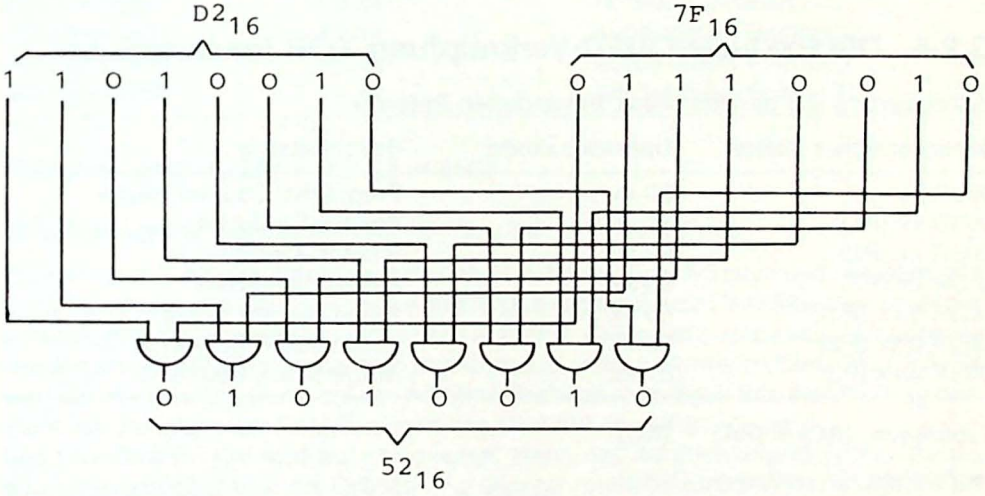


Bild 3.2.3.1: Schaltung für die Simulation des AND-Befehls

3.2.4 Die ODER-Verknüpfung OR (or)

Bezüglich der Adressierung gelten die gleichen Regeln wie bei der UND-Verknüpfung.

Mnemonischer Befehl	Operationskode	Beschreibung
OR xx	D8 xx	Programm Counter relativ
OR xx (P1)	D9 xx	Pointer 1 indiziert
OR xx (P2)	DA xx	Pointer 2 indiziert
OR xx (P3)	DB xx	Pointer 3 indiziert
OR @xx (P1)	DD xx	Autoindiziert Pointer 1
OR @xx (P2)	DE xx	Autoindiziert Pointer 2
OR @xx (P3)	DF xx	Autoindiziert Pointer 3

Operation: **(AC)** $\vee$ **(EA)** $\rightarrow$ **(AC)**

$\vee$ Zeichen für logisches ODER

Der Inhalt des Akkumulators wird mit dem Inhalt der effektiven Adresse ODER-verknüpft. Das Ergebnis steht im Akku, der alte Inhalt der angesprochenen Speicherstelle wird nicht verändert. Auch hier wird die Verknüpfung der beiden Operanden AC und EA Bit für Bit durchgeführt. Es gelten die allgemeinen Regeln der logischen ODER-Verknüpfung. Mit Hilfe dieses Befehls können gezielt einzelne Bits eines Datenbytes auf »1« gesetzt werden.

3.2.5 Die Exklusiv-ODER-Verknüpfung XOR (exclusive or)

Adressierung wie bei den bisher behandelten Befehlen.

Mnemonischer Befehl	Operationskode	Beschreibung
XOR xx	E0 xx	Programm Counter relativ
XOR xx (P1)	E1 xx	Pointer 1 indiziert
XOR xx (P2)	E2 xx	Pointer 2 indiziert
XOR xx (P3)	E3 xx	Pointer 3 indiziert
XOR @xx (P1)	E5 xx	Autoindiziert Pointer 1
XOR @xx (P2)	E6 xx	Autoindiziert Pointer 2
XOR @xx (P3)	E7 xx	Autoindiziert Pointer 3

Operation: **(AC)** $\veebar$ **(EA)** $\rightarrow$ **(AC)**

$\veebar$ Zeichen für exklusives ODER

Der Inhalt des Akkumulators wird mit dem Inhalt der effektiven Adresse exklusiv-ODER-verknüpft. Das Ergebnis steht im Akku. Der alte Inhalt der effektiven Adresse wird nicht beeinflußt.

Mit Hilfe des XOR-Befehls lassen sich einzelne Bits bzw. ein ganzes Byte komplementieren. Aus »1« wird »0« und umgekehrt. Will man ein ganzes Byte komplementieren, so muß eine Exklusiv-ODER-Verknüpfung mit FF_{16} (= 11111111_2) durchgeführt werden.

Annahme: (AC) = $D5_{16}$ = 11010101_2
Pointer 1 = 1000_{16}, (1000_{16}) = FF_{16}, (1001_{16}) = 80_{16}

Nach Ausführung der Instruktion 'XOR 00 (P1)' steht im Akku der komplementierte (Einerkomplement) Wert $2A_{16}$des alten Inhaltes. Anhand der Regeln der Exklusiv-ODER-Verknüpfung läßt sich das Ergebnis leicht nachprüfen.
Mit dem Befehl 'XOR 01 (P1)' wird beim gerade aktuellen Akkuinhalt das höchstwertige Bit komplementiert.

3.2.6 Der Additions-Befehl ADD (add)

Adressierung wie bei den bisher behandelten Befehlen.

Mnemonischer Befehl	Operationskode	Beschreibung
ADD xx	F0 xx	Programm Counter relativ
ADD xx (P1)	F1 xx	Pointer 1 indiziert
ADD xx (P2)	F2 xx	Pointer 2 indiziert
ADD xx (P3)	F3 xx	Pointer 3 indiziert
ADD @xx (P1)	F5 xx	Autoindiziert Pointer 1
ADD @xx (P2)	F6 xx	Autoindiziert Pointer 2
ADD @xx (P3)	F7 xx	Autoindiziert Pointer 3

Operation: **(AC) + (EA) + (CY/L) → (AC)**

Im Statusregister werden die Bits »CY/L« und »OV« beeinflußt.

Zum Inhalt des Akkumulators wird der Inhalt der effektiven Adresse und der Wert des CY/L-Bits (kann »0« oder »1« sein) addiert. Das Ergebnis steht im Akku. Der Inhalt der adressierten Speicherstelle wird nicht verändert. Die Inhalte von Akku und Adresse werden als achtstellige Dualzahlen (siehe dazu Kapitel 1.3) interpretiert. Das CY/L-Bit wird auf »1« gesetzt, wenn bei der Addition der höchstwertigen Bits ein Übertrag entsteht. Ist das nicht der Fall, so nimmt das CY/L-Bit den Wert »0« an.
Das Overflow-Bit OV wird auf »1« gesetzt, wenn das Additionsergebnis die Grenze 80_{16} unterschreitet bzw. die Grenze $7F_{16}$ überschreitet. Im anderen Fall ist dieses Bit »0«.

Dazu ein paar Beispiele. Bei allen wird vorausgesetzt, daß das CY/L- und das OV-Bit zu Beginn den Wert »0« haben.

Annahme: Pointer 1 = $AF00_{16}$, Pointer 2 = 2000_{16}
($AF00_{16}$) = 25_{16}, ($AF01_{16}$) = 29_{16}, ($AF02_{16}$) = E_{16}
($AF03_{16}$) = 75_{16}, ($AF04_{16}$) = $A0_{16}$, ($AF05_{16}$) = 00_{16}

Lassen wir den Mikroprozessor wieder ein paar Instruktionen abarbeiten und sehen wir uns an, was dabei geschieht.

Beispiel 1:

```
LD 00(P1)     ; (AC) erhält den Wert 25(16)
ADD 01(P1)    ; Addition des Inhaltes von Adresse AF01(16)
              ; zum Inhalt des Akkus, Ergebnis im Akku
ST 00(P2)     ; Ergebnis 4E(16) nach Adresse 2000(16) schreiben
```

CY/L und OV bleiben »0«. Anhand der Regeln aus Kapitel 1.3.6 läßt sich das Resultat leicht nachprüfen.

Beispiel 2:

```
LD 00(P1)     ; 29(16) → (AC)
ADD 02(P1)    ; 29(16)+6E(16)+(CY/L) → (AC)
ST 00(P2)     ; 93(16) → (2000(16))
```

CY/L bleibt »0«, das OV-Bit wird »1«, weil die Grenze $7F_{16}$ überschritten wurde.

Beispiel 3:

```
LD 03(P1)     ; 75(16) → (AC)
ADD 04(P1)    ; 75(16)+A0(16)+(CY/L) → (AC)
ST 00(P2)     ; 15(16) → (2000(16)), CY/L = »1«
ADD 05(P1)    ; 15(16)+00(16)+01(16) → (AC), CY/L wird »0«
ST 01(P2)     ; 16(16) → (2000(16))
```

OV wird erst »1« und bei der zweiten Addition wieder »0«. CY/L ebenfalls erst »1« und danach wieder »0«. Sehen wir uns das letzte Beispiel auf Dualzahlenebene an.

Befehl	Sedezimalzahl	CY/L	Dualzahl	OV
	75	0	01110101	0
ADD			0	
	A0	0	10100000	0
	15	1	00010101	1
ADD			1	
	00	1	00000000	1
	16	0	00010110	0

Hierbei wird klar, worauf es ankommt. Will man sicherstellen, daß bei einer Addition nicht fälschlicherweise das gesetzte CY/L-Bit addiert wird, so muß man es gezielt mit dem Befehl CCL löschen. In späteren Beispielen wird darauf noch näher eingegangen.

3.2.7 Befehl für die dezimale Addition DAD (decimal add)

Adressierung wie bisher.

Mnemonischer Befehl	Operationskode	Beschreibung
DAD XX	E8 XX	Programm Counter relativ
DAD XX(P1)	E9 XX	Pointer 1 indiziert
DAD XX(P2)	EA XX	Pointer 2 indiziert
DAD XX(P3)	EB XX	Pointer 3 indiziert
DAD @XX(P1)	ED XX	Autoindiziert Pointer 1
DAD @XX(P2)	EE XX	Autoindiziert Pointer 2
DAD @XX(P3)	EF XX	Autoindiziert Pointer 3

Operation: $\mathbf{(AC)_{10} + (EA)_{10} + (CY/L) \rightarrow (AC)}$

Im Statusregister wird das CY/L-Bit beeinflußt.

Der Mikroprozessor interpretiert die Inhalte von Akkumulator und effektiver Adresse als zweistellige BCD-Zahlen.
Der Inhalt der angesprochenen Speicherstelle und der Wert des CY/L werden zum Inhalt des Akkus addiert. Das Ergebnis, eine zweistellige BCD dargestellte Zahl, steht im Akku. Der alte Inhalt der effektiven Adresse bleibt erhalten. Das OV-Bit wird bei Ausführung dieser Instruktion nicht beeinflußt.
Tritt bei der Addition bei den höchstwertigen Bits ein Übertrag auf, so wird das CY/L-Bit auf »1« gesetzt; sonst auf »0«.
Bei den nachfolgenden Beispielen ist wieder vorausgesetzt, daß das CY/L-Bit vor Befehlsausführung den Wert »0« hat. Die Pointerwerte übernehmen wir aus Kapitel 3.2.6.

Beispiel 1:

```
LD 00(P1)     ;  25(10) → (AC)
DAD 01(P1)    ;  25(10)+29(10)+CY/L → (AC), CY/L bleibt »0«
ST 00(P2)     ;  54(10) → (2000(16))
```

Beispiel 2:

```
LD 00(P1)     ;  25(10) → (AC)
DAD 03(P1)    ;  25(10)+75(10)+CY/L → (AC)(10), CY/L wird »1«
ST 00(P2)     ;  00 → (2000(16))
```

In Beispiel 2 signalisiert das gesetzte CY/L-Bit, daß das Ergebnis nicht 0 sondern 100_{10} ist, denn es hat ein Übertrag in die nächsthöhere BCD-Stelle (hier die 100-er Stelle) stattgefunden.
In einem Programm kann der Zustand des CY/L z. B. für die richtige Anzeige des Ergebnisses benutzt werden.

3.2.8 Der Komplement-Additions-Befehl CAD (complement and add)

Adressierung wie bisher.

Mnemonischer Befehl	Operationskode	Beschreibung
CAD XX	F8 XX	Programm Counter relativ
CAD XX(P1)	F9 XX	Pointer 1 indiziert
CAD XX(P2)	FA XX	Pointer 2 indiziert
CAD XX(P3)	FB XX	Pointer 3 indiziert
CAD @XX(P1)	FD XX	Autoindiziert Pointer 1
CAD @XX(P2)	FE XX	Autoindiziert Pointer 2
CAD @XX(P3)	FF XX	Autoindiziert Pointer 3

Operation: **(AC)** + **($\overline{EA}$)** + **(CY/L)** → **(AC)**

Im Statusregister werden CY/L und OV geändert.

Wie beim ADD-Befehl werden auch hier die Inhalte von Akku und effektiver Adresse als acht Bit breite Dualzahlen interpretiert. Die Instruktion CAD bewirkt, daß zum Inhalt des Akkumulators das Einerkomplement vom Inhalt der adressierten Speicherstelle dargestellt durch den Querstrich) und der Wert des CY/L-Bits addiert wird.
ritt an der höchstwertigen Stelle ein Übertrag auf, so wird das CY/L-Bit auf »1« esetzt. Im anderen Fall erhält es den Wert »0«. Das OV-Bit wird immer dann »1«, wenn das Ergebnis der Addition die Grenze 80_{16} unterschreitet oder die Schranke $7F_{16}$ überschreitet.
Wie aus Kapitel 1.3.6 bekannt ist, wird die duale (binäre) Subtraktion ausgeführt, indem zum Minuenden das Zweierkomplement des Subtrahenden addiert wird.
Beim CAD-Befehl wird jedoch nur das Einerkomplement addiert und dieses ist immer um genau eins kleiner als das Zweierkomplement einer Zahl. Wollen wir eine Subtraktion richtig ausführen, so müssen wir zum Ergebnis demnach eins addieren. Setzen wir vor der Ausführung der Instruktion CAD das CY/L-Bit auf »1«, so wird dieses »eins addieren« automatisch mit CAD ausgeführt, denn der Inhalt des CY/L wird ja ebenfalls addiert. Mit dem Befehl SCL können wir CY/L gezielt auf »1« setzen.

Dazu sehen wir uns zwei Beispiele an.

Beispiel 1: CY/L = 0, Pointer 1 = 1000_{16}
(1000_{16}) = $1A_{16}$, (1001_{16}) = $0A_{16}$

```
LD 00(P1)     ;  1A16 → (AC)
CAD 01(P1)    ;  1A16+0A16+CY/L → (AC), CY/L wird »1«
ST 02(P1)     ;  0F16 → (100216)
```

Das Ergebnis ist um eins zu klein.

Beispiel 2:
```
SCL          ; 1 → (CY/L)
LD 00(P1)    ; 1A16 → (AC)
CAD 01(P1)   ; 1A16+0A̅16+1 → (AC), CY/L bleibt »1«
ST 02(P1)    ; 1016 → (100216)
```

Der kleine Trick bewirkt, daß das Ergebnis richtig ist.
In Kapitel 4.4 und 4.5 ist die Anwendung dieser Instruktion in einem Programm zu sehen.

3.3 Inkrement- und Dekrement Befehle

Mit diesen Befehlen kann der Inhalt einer Speicherstelle um eins inkrementiert bzw. dekrementiert werden. Es lassen sich damit z. B. Schleifenzähler realisieren.

3.3.1 Inkrementiere und Lade Befehl ILD (increment and load)

Es ist keine autoindizierte Adressierung möglich. Bezüglich des Displacements gilt das bisher gesagte.

Mnemonischer Befehl	Operationskode	Beschreibung
ILD XX	A8 XX	Programm Counter relativ
ILD XX(P1)	A9 XX	Pointer 1 indiziert
ILD XX(P2)	AA XX	Pointer 2 indiziert
ILD XX(P3)	AB XX	Pointer 3 indiziert

Operation: **(EA) + 1 → (EA), (AC)**

Statusbits werden nicht beeinflußt.

Der Inhalt der effektiven Adresse wird um eins erhöht und der inkrementierte Wert wird in den Akku geladen.

Annahme: Pointer 1 = 1000_{16}, Pointer 2 = 3000_{16}
(1000_{16}) = $2E_{16}$, (3000_{16}) = FF_{16}

Läßt man den SCMP folgende Befehle abarbeiten, so werden der Inhalt der effektiven Adresse und der Inhalt des Akkumulators wie rechts aufgeführt verändert.

```
ILD 00(P1)   ; 2F16 → (AC), 2F16 → (EA)
ILD 00(P1)   ; 3016 → (AC), 3016 → (EA)
ILD 00(P2)   ; 00 → (AC), 00 → (EA)
ILD 00(P2)   ; 0116 → (AC), 0116 → (EA)
```

Dieser Befehl ist recht einfach, so daß weitere Beispiele erst im Zusammenhang mit den Sprungbefehlen gebracht werden.

3.3.2 Dekrement und Lade Befehl DLD (decrement and load)

Adressierung und Displacement wie beim ILD-Befehl.

Mnemonischer Befehl	Operationskode	Beschreibung
DLD XX	B8 XX	Programm Counter relativ
DLD XX(P1)	B9 XX	Pointer 1 indiziert
DLD XX(P2)	BA XX	Pointer 2 indiziert
DLD XX(P3)	BB XX	Pointer 3 indiziert

Operation: **(EA) – 1 → (EA), (AC)**

Es werden keine Statusbits beeinflußt.

Im Gegensatz zum ILD-Befehl wird hier der Inhalt der effektiven Adresse um eins vermindert und der dekrementierte Wert in den Akkumulator geladen.

Legen wir die Pointerwerte und Adreßinhalte von Kapitel 3.3.1 zu Grunde, so ergibt sich bei der Ausführung nachfolgender Instruktionen das rechts stehende Ergebnis.

DLD 00(P1) ; $2D_{16} \rightarrow (AC)$, $2D_{16} \rightarrow (EA)$
DLD 00(P1) ; $2C_{16} \rightarrow (AC)$, $2C_{16} \rightarrow (EA)$
DLD 00(P2) ; $FE_{16} \rightarrow (AC)$, $FE_{16} \rightarrow (EA)$

Weitere Beispiele auch hier bei der Behandlung der Sprungbefehle in Kapitel 3.5

3.4 Immediate (unmittelbare) Befehle

Auch diese Befehlsgruppe besteht aus lauter Zwei-Byte-Instruktionen. Das erste Byte gibt den Operationskode an und das zweite Byte stellt die Daten, die verarbeitet werden sollen, dar. D. h., bei allen immediate Befehlen wird keine andere Speicherstelle als die unmittelbar hinter dem Operationskode stehende adressiert.

3.4.1 Lade unmittelbar Befehl LDI (load immediate)

Mnemonischer Befehl: LDI XX
Operationskode : C4 XX

Operation: **XX → (AC)**

Das hinter dem Befehl stehende Datenbyte (XX) wird in den Akku geladen. Der ursprüngliche Inhalt des Akkus geht verloren.
Die Statusbits werden nicht verändert.

Wird die Instruktion LDI FF_{16} vom SCMP ausgeführt, so erhält der Akku nach Ausführung den Wert FF_{16}. Man kann damit Konstanten die bei der Programmerstellung schon bekannt sind direkt laden und braucht damit für sie nicht irgendwo ein Label vorsehen.

3.4.2 Addiere unmittelbar Befehl ADI (add immediate)

Mnemonischer Befehl: ADI XX
Operationskode : F4 XX

Operation: **(AC) + XX + (CY/L) → (AC)**

Zum Inhalt des Akkumulator wird das Datenbyte (XX) und der Inhalt des CY/L addiert. Das Ergebnis steht im Akku. Wird das Resultat kleiner als 80_{16} oder größer als $7F_{16}$, so wird das OV-Bit auf »1« gesetzt. Das CY/L-Bit wird »1«, wenn bei der Addition an der höchstwertigen Stelle ein Übertrag auftritt (siehe auch 3.2.6). Im anderen Fall wird es »0«.

Beispiel:

```
CCL       ; 0 → (CY/L)
LDI 05    ; 05₁₆ → (AC)
ADI 38    ; 05₁₆+38₁₆+0=3D₁₆ → (AC)
```

CY/L- und OV-Bit werden bei obigem Beispiel nicht verändert.

Beispiel:

```
SCL       ; 1 → (CY/L)
LDI 00    ; 0 → (AC)
ADI 00    ; 0 + 0 + 1 = 01₁₆ → (AC), CY/L wird »0«
```

In diesem Beispiel wird deutlich, daß das CY/L-Bit immer addiert wird. Man muß sich darum beim Programmieren über den Wert dieses Bits im Klaren sein, wenn bei der Programmabarbeitung keine Überraschungen bezüglich des Ergebnisses auftreten sollen. Es gelten die Regeln der speicherbezogenen Addition.

3.4.3 Unmittelbar dezimal Addition DAI (decimal add immediate)

Mnemonischer Befehl: DAI XX
Operationskode : EC XX

Operation: **$(AC)_{10} + XX_{10} + (CY/L) \rightarrow (AC)$**

Das OV-Bit wird bei dieser Instruktion nicht beeinflußt. Das CY/L-Bit wird gesetzt, wenn an der höchstwertigen BCD-Stelle bei den höchstwertigen Bits ein Übertrag auftritt. Bei der dezimalen Addition wird das Datenbyte als zweistellige BCD-Zahl verarbeitet.

Beispiel:

```
CCL          ;0 → (CY/L)
LDI 13       ;13₁₀ → (AC)
DAI 74       ;13₁₀ + 74₁₀ + 0 = 87₁₀ → (AC), CY/L bleibt »0«
```

3.4.4 Komplementiere und addiere unmittelbar CAI (complement and add immediate)

Mnemonischer Befehl: CAI XX
Operationskode : FC XX

Operation: **(AC) + $\overline{XX}$ + (CY/L) → (AC)**

Es werden das CY/L-Bit und das OV-Bit beeinflußt. Dabei gelten die gleichen Regeln wie in Kapitel 3.2.8 bereits beschrieben. Bei der Instruktion CAI wird zum Inhalt des Akku das Einerkomplement des Datenbytes und der Wert des CY/L-Bits addiert.

Das Ergebnis steht wiederum im Akkumulator. Will man mit diesem Befehl die Subtraktion (AC) – Datenbyte durchführen, so muß man vorher das CY/L-Bit auf »1« setzen (siehe dazu auch Kapitel 3.2.8). Im nachfolgenden Beispiel wird das demonstriert.

```
CCL          ;  1 → (CY/L)
LDI 5C       ;  5C₁₆ → (AC)
CAI 32       ;  5C₁₆ + 32̄₁₆ + 1 = 2A₁₆ → (AC), CY/L wird »1«
```

Mit Hilfe des Befehls CAI können innerhalb eines Programms Abfragen auf Zahlenbereiche z. B. größer, kleiner oder gleich durchgeführt werden. Zusammen mit den Sprungbefehlen folgen weitere Beispiele.

3.4.5 Unmittelbare UND-Verknüpfung ANI (and immediate)

Mnemonischer Befehl: ANI XX
Operationskode : D4 XX

Operation: **(AC) ∧ XX → (AC)**

Der Inhalt des Akkus wird mit dem Datenbyte (XX) UND-verknüpft und das Ergebnis im Akku abgelegt. Damit lassen sich z. B. aus einem Datenstrom durch die UND-Verknüpfung mit einer Maske (Datenbyte) bestimmte Bits auf »1« abfragen, bzw. einzelne Bits oder Bitmuster in einem Byte löschen.

Beispiel: Aus einem Datenstrom sollen alle Bytes, bei denen das höchstwertige Bit gesetzt ist (Zahl negativ), herausgefunden werden.

```
⋮
LD 00(P1)      ; aktuelles Datenbyte lesen
ANI 80         ; mit 80₁₆ UND-verknüpfen, Ergebnis in Akku
JZ POSITIV     ; Sprung nach POSITIV, wenn das Ergebnis der
⋮              ; UND-Verknüpfung Null ist (Bit 2⁷ = 0).
·              ; Im anderen Fall wird mit der nächsten Instruktion
·              ; fortgefahren.
```

3.4.6 Unmittelbare ODER-Verknüpfung ORI (or immediate)

Mnemonischer Befehl: ORI XX
Operationskode : DC XX

Operation: **(AC) ∨ XX → (AC)**

Der Inhalt des Akkus wird mit dem Datenbyte ODER-verknüpft und das Resultat im Akkumulator abgelegt. Es gelten auch hier die allgemeinen Regeln der ODER-Verknüpfung. Mit Hilfe dieser Instruktion lassen sich einzelne Bits oder Bitmuster in einem Byte setzen. Dies ist möglich ohne bereits vorhandene Bitkombinationen zu ändern. Bei einer ASCII-Zeichenübertragung könnte damit das Bit 7 (wird nicht im ASCII-Zeichensatz verwendet) als Paritätsbit gesetzt bzw. nicht gesetzt werden.
ORI 80 setzt unabhängig vom alten Akkumulatorinhalt zusätzlich das Bit 7 in dem im Akku stehenden Datenbyte.

3.4.7 Unmittelbare EXklusiv-ODER-Verknüpfung XRI (exclusive or immediate)

Mnemonischer Befehl: XRI XX
Operationskode : E4 XX

Operation: **(AC) ⩛ XX → (AC)**

Der Inhalt des Akkus wird mit dem Datenbyte (XX) exklusiv-ODER-verknüpft und das Ergebnis im Akkumulator abgelegt.
Es gelten die Regeln für die Exklusiv-ODER-Verknüpfung. Auch hier, wie bei allen logischen Verknüpfungsbefehlen, wird die Operation Bit für Bit durchgeführt.
Mit dem XRI-Befehl können einzelne Bits oder Bitmuster in einem Byte komplementiert werden.
Nehmen wir an es ist erforderlich die beiden höchstwertigen Bits (Bit 6 und 7) zu komplementieren. Die Instruktion

```
⋮              ; (AC) = aktuelles Datenbyte
XRI CO         ; (AC) ⩛ CO₁₆ → (AC)
⋮
```

führt dies durch. Z. B. wird aus $7A_{16}$ der Wert BA_{16}.

3.5 Die Sprungbefehle

Erst die Gruppe der Sprungbefehle macht den Mikroprozessor zu dem, was er ist. Damit ist es möglich, Abfragen mit entsprechenden Verzweigungen zu realisieren. Einige Mikroprozessoren verfügen über eine große Anzahl von Sprungbefehlen, die mit Abfragen (z. B. größer, kleiner, gleich u. a.) kombiniert sind. Unser SCMP besitzt insgesamt nur vier Sprungbefehle. Drei davon sind bedingte Sprünge. Die geringe Anzahl von Sprungbefehlen mit Bedingung ist kein großer Nachteil. Man muß in diesem Fall mehr in die »Vorarbeit« (aufbereiten der Daten) stecken.
Allen Sprüngen gemeinsam ist, daß bei der Ausführung der Programm Counter entsprechend dem angegebenen Displacement »XX« verändert wird. Hier jedoch heißt es aufpassen. Der SCMP hat die Eigenart, daß er nach Ausführung eines Sprungs den Programm Counter sofort um eins erhöht. Bei der Berechnung des Offsets ist diese Tatsache zu berücksichtigen.
Doch wie kommt es zu diesem Verhalten.
Wird ein Befehl im Lesezyklus ins Instruktionsregister geholt, so wird anschließend gleich der Programm Counter inkrementiert. Bei einem Zwei-Byte-Befehl liest der Prozessor jetzt das zweite Byte des Befehls. Zu diesem Programm Counter Stand, dort wo das Displacement steht, addiert der SCMP jetzt den Wert desselben und erhöht danach das Ergebnis noch einmal um eins. So viel zum inneren Ablauf.
Für die Programm Counter relativen Sprünge gilt:

Displacement »XX« = Zieladresse − Absprungadresse − 2

Absprungadresse ist die Adresse an der der Sprungbefehl selbst steht. Zieladresse ist die Adresse des Labels das angesprungen werden soll. Außer den PC-relativen Sprüngen gibt es noch die Pointer indizierten Sprünge. Bei diesen steht im durch den Befehl ausgewählten Pointer das Sprungziel. Das als zweites Byte in der Sprunginstruktion stehende Displacement wird zum Pointerinhalt addiert (subtrahiert). Das Ergebnis ergibt den neuen Wert des Programm Counters, der jedoch gleich um eins erhöht wird. Von der damit adressierten Speicherstelle wird die nächste Instruktion geholt. Bei der Angabe des Displacements muß dieser Umstand berücksichtigt werden.

3.5.1 Unbedingter Sprung JMP (jump)

Dieser Sprung wird immer ausgeführt, da er von keiner Bedingung abhängig ist.

Mnemonischer Befehl	Operationskode	Beschreibung
JMP XX	90 XX	Programm Counter relativ
JMP XX(P1)	91 XX	Pointer 1 indiziert
JMP XX(P2)	92 XX	Pointer 2 indiziert
JMP XX(P3)	93 XX	Pointer 3 indiziert

Annahme: Pointer 1 = 2000_{16}, Pointer 2 = $A120_{16}$

Adresse	Befehl		Funktion
0100_{16}	JMP 10	;	Der nächste Befehl wird von Adresse
⋮	⋮	;	0112_{16} geholt, d. h. das Programm wird
		;	an dieser Stelle fortgesetzt.
$02A0_{16}$	JMP 00(P1)	;	Das Programm wird an der Adresse 2001_{16}
⋮	⋮	;	fortgesetzt.
$02A2_{16}$	JMP FF(P1)	;	Programmfortsetzung an Adresse 2000_{16}
⋮	⋮		
.	.		

3.5.2 Sprung bei positivem Akkuinhalt JP (jump if positive)

Dieser bedingte Sprung wird nur ausgeführt, wenn der Inhalt des Akkumulators positiv ist. Das höchstwertige Bit ist in diesem Fall 0. Dies gilt für den Wertebereich 00_{16} – $7F_{16}$.

Mnemonischer Befehl	Operationskode	Beschreibung
JP XX	94 XX	Programm Counter relativ
JP XX(P1)	95 XX	Pointer 1 indiziert
JP XX(P2)	96 XX	Pointer 2 indiziert
JP XX(P3)	97 XX	Pointer 3 indiziert

Für nachfolgende Beispiele werden die Pointerinhalte aus 3.5.1 angenommen.

Adresse	Befehl		Funktion
0100_{16}	LDI 3A	;	$3A_{16} \rightarrow$ (AC)
0102_{16}	JP 10	;	Das Programm wird an Adresse 0112_{16}
		;	fortgesetzt.
0104_{16}	LDI B2	;	$B2_{16} \rightarrow$ (AC)
0106_{16}	JP 01(P1)	;	kein Sprung, da (AC) nicht positiv
0108_{16}	LDI 00	;	hier wird das Programm fortgesetzt, $0 \rightarrow$ (AC)
$010A_{16}$	JMP FF(P2)	;	der nächste Befehl wird von Adresse $A120_{16}$
		;	geholt und das Programm dort fortgesetzt

3.5.3 Sprung bei Akkuinhalt Null JZ (jump if zero)

Der Sprung wird nur ausgeführt, wenn der Akkuinhalt Null (00) ist. Im anderen Fall wird das Programm mit dem nächsten Befehl weiter bearbeitet.

Mnemonischer Befehl	Operationskode	Beschreibung
JZ XX	98 XX	Programm Counter relativ
JZ XX(P1)	99 XX	Pointer 1 indiziert
JZ XX(P2)	9A XX	Pointer 2 indiziert
JZ XX(P3)	9B XX	Pointer 3 indiziert

Beispiel:

```
LD 01(P1)     ; (2001₁₆) → (AC)
JZ ZERO       ; Sprung nach ZERO, wenn (AC) = 00 sonst
ST 02(P1)     ; (AC) → (2002₁₆)
```

Beispiel einer Abfrage auf einen bestimmten Zahlenwert (hier 21_{16}).

```
LD 00(P1)     ; (2000₁₆) → (AC)
SCL           ; 1 → (CY/L)
CAI 21        ; (AC) + 21̄₁₆ + 01 → (AC)
JZ IST21      ; Sprung nach IST21, wenn (2000₁₆) = 21₁₆ war.
:             ; Im anderen Fall hier Fortsetzung des Programms.
```

3.5.4 Sprung bei Akkuinhalt ungleich Null JNZ (jump if not zero)

Dieser Sprung wird immer ausgeführt, wenn der Inhalt des Akkumulators nicht Null ist.

Mnemonischer Befehl	Operationskode	Beschreibung
JNZ XX	9C XX	Programm Counter relativ
JNZ XX(P1)	9D XX	Pointer 1 indiziert
JNZ XX(P2)	9E XX	Pointer 2 indiziert
JNZ XX(P3)	9F XX	Pointer 3 indiziert

Die in Kapitel 3.5.3 gezeigte Abfrage auf den Zahlenwert 21_{16} kann man auch mit dem Befehl JNZ durchführen. Der Unterschied besteht darin, daß hierbei das Programm unmittelbar hinter dem Sprungbefehl weiter abgearbeitet wird, wenn das geladene Byte den Wert 21_{16} hat.

```
LD 00(P1)     ; (2000₁₆) → (AC)
SCL           ; 1 → (CY/L)
CAI 21        ; vom Akkuinhalt 21₁₆ abziehen
JNZ NOT21     ; Weiter bei NOT21, wenn das Datenbyte nicht
              ; den Wert 21₁₆ hat,
⋮             ; sonst hier im Programm weiter
```

Man kann Abfragen in den meisten Fällen auf mehrere Arten durchführen. In der Regel gibt es jedoch eine Lösung die für die gestellte Aufgabe optimal ist. Dazu gleich noch eine Anmerkung.
Optimal kann in einem Fall »zeitoptimal« und in einem anderen Fall »speicherplatzoptimal« sein. In der Praxis ist nach den entsprechenden Vorgaben zu verfahren.

3.6 Die Pointerregister Austauschbefehle

Unter bestimmten Bedingungen kann man auch diese Instruktionsgruppe als Sprungbefehle benutzen.
Bis auf eine Ausnahme, die in 3.6.3 beschrieben ist, wird bei allen diesen Befehlen der Akkumulatorinhalt mit dem Inhalt einer Pointerregisterhälfte ausgetauscht. Im Falle des Austausches zwischen Akku und Programm Counter (Pointer 0) hat dies einen Sprung zur Folge. Das Sprungziel wird durch den neuen PC-Inhalt definiert.

3.6.1 Austausch LOB eines Pointers mit Akku XPAL (exchange pointer low with accu)

LOB steht für »Low Order Byte« was so viel wie niederwertigstes Byte bedeutet.
Bei diesen Instruktionen wird das LOB des PC oder eines anderen Pointers mit dem Inhalt des Akkumulators ausgetauscht. Wendet man die Anweisung auf den Programmzähler an, so können damit Sprünge in einem Bereich PC plus 00_{16} bis FF_{16} ausgeführt werden.

Mnemonischer Befehl	Operationskode	Funktion
XPAL 0	30	PC(LOB) ⟷ (AC)
XPAL 1	31	P1(LOB) ⟷ (AC)
XPAL 2	32	P2(LOB) ⟷ (AC)
XPAL 3	33	P3(LOB) ⟷ (AC)

Beispiel 1: Pointer 1 = $1A2F_{16}$

```
LDI 00        ; 0 → (AC)
XPAL 1        ; 2F₁₆ → (AC), 00 → (LOB(P1))
```

Pointer 1 hat jetzt den Wert $1A00_{16}$.

Beispiel 2:

Adresse	Befehl		Funktion
1000_{16}	LDI 0F	;	$0F_{16} \rightarrow$ (AC)
1002_{16}	XPAL 0	;	$0F_{16} \rightarrow$ (LOB(PC)), XX $\rightarrow$ (AC)
⋮	⋮		
1010_{16}	nächster Befehl der ausgeführt wird		

Man kann damit Programmverzweigungen abhängig von bestimmten Variablen, zum Beispiel dem Wert einer Funktionstaste, realisieren.

3.6.2 Austausch HOB eines Pointers mit Akku XPAH (exchange pointer low with accu)

HOB steht für »High Order Byte« und bedeutet, daß es sich hierbei um das höherwertige Byte handelt.
Bei der Befehlsausführung wird das HOB des Programm Counters bzw. eines anderen Pointers mit dem Inhalt des Akkumulators ausgetauscht. Auf den Programm Counter angewendet, lassen sich damit Sprünge über die Seitengrenzen (Pages) hinweg durchführen.

Mnemonischer Befehl	Operationskode	Funktion
XPAH 0	34	PC(HOB) $\leftrightarrow$ (AC)
XPAH 1	35	P1(HOB) $\leftrightarrow$ (AC)
XPAH 2	36	P2(HOB) $\leftrightarrow$ (AC)
XPAH 3	37	P3(HOB) $\leftrightarrow$ (AC)

Beispiel 1: Pointer 3 = $F200_{16}$

LDI 00	;	0 $\rightarrow$ (AC)
XPAL 3	;	0 $\rightarrow$ (LOB(P3)), 0 $\rightarrow$ (AC)
LDI 20	;	$20_{16} \rightarrow$ (AC)
XPAH 3	;	$20_{16} \rightarrow$ (HOB(P3)), $F2_{16} \rightarrow$ (AC)

Nach Ausführung obiger Anweisungen hat der Pointer 3 den neuen Inhalt 2000_{16}.

Beispiel 2:

Adresse	Befehl		Funktion
0010_{16}	LDI A1	;	$A1_{16} \rightarrow$ (AC)
0012_{16}	XPAH 0	;	$A1_{16} \rightarrow$ (HOB(PC)), 0 $\rightarrow$ (AC)
⋮	⋮		
$A113_{16}$	hier wird der nächste Befehl geholt und abgearbeitet.		

Der Programm Counter wird auf $A112_{16}$ gesetzt um eins erhöht und demnach das Programm bei Adresse $A113_{16}$ weiter bearbeitet.
Es hat damit ein Wechsel von »Page 0« zur »Page A« stattgefunden.

3.6.3 PC-Inhalt und Pointer-Inhalt austauschen XPPC (exchange pointer and programm counter)

Der Befehl mit dem Unterprogramme aufgerufen werden können bewirkt, daß der aktuelle Programmzählerinhalt und der Inhalt des angesprochenen Pointers ausgetauscht werden. Nach Ausführung der Instruktion wird anschließend der neue Pointerinhalt gleich um eins inkrementiert. Aus diesem Grund muß die um eins verminderte Startadresse eines Unterprogramms in den entsprechenden Pointer geladen werden, bzw. der erste Befehl in der Subroutine muß ein Leerbefehl (NOP) sein.

Mnemonischer Befehl	Operationskode	Funktion
XPPC 0	3C	(PC) ⟷ (PC)
XPPC 1	3D	(PC) ⟷ (P1)
XPPC 2	3E	(PC) ⟷ (P2)
XPPC 3	3F	(PC) ⟷ (P3)

In einem Beispiel wollen wir uns anschauen wie normalerweise ein Unterprogrammaufruf und der Rücksprung aus diesem durchgeführt wird. Es soll ein Programm welches an Adresse $1AF7_{16}$ beginnt aufgerufen werden. In unserem UP (Unterprogramm) wird nur der Akku mit dem Wert FF_{16} geladen.

Adresse	Inhalt	Befehl		Funktion
0000_{16}	08	NOP	;	Leerbefehl
0001_{16}	C4F6	LDI F6	;	$F6_{16}$ → (AC)
0003_{16}	33	XPAL 3	;	$F6_{16}$ → (LOB(P3))
0004_{16}	C41A	LDI 1A	;	$1A_{16}$ → (AC)
0006_{16}	37	XPAH 3	;	$1A_{16}$ → (HOB(P3))
0007_{16}	3F	XPPC 3	;	$1AF6_{16}$ → (PC), 0007_{16} → (P3)
0008_{16}	:	:	;	nach Rücksprung aus dem UP wird
.	.	.	;	hier das Hauptprogramm fortgesetzt.

Unterprogramm:

Adresse	Inhalt	Befehl		Funktion
$1AF7_{16}$	C4FF ANF:	LDI FF	;	FF_{16} → (AC)
$1AF9_{16}$	3F	XPPC 3	;	0007_{16} → (PC), $1AF9_{16}$ → (P3)
$1AFA_{16}$	90FB	JMP ANF	;	Sprung an UP-Anfang

Der Sprung an den Anfang des Unterprogramms am Ende dieses Programms ist notwendig damit bei erneutem Aufruf des UP dieses ausgeführt werden kann. In Pointer 3 steht nach dem Rücksprung nicht mehr die Anfangsadresse sondern die Adresse des Rücksprungs (der PC wurde ja im UP verändert). Beim nächsten Aufruf wird an Adresse $1AF9_{16}$ gesprungen und der Befehl von $1AFA_{16}$ geholt. Der dort stehende Sprung zum Anfang der Routine ermöglicht einen ordnungsgemäßen Ablauf. Man kann sich den Sprung sparen, wenn man vor jedem Aufruf die Anfangsadresse wieder in den entsprechenden Pointer lädt. Ebenso ist möglich und oft aus Gründen der Übersichtlichkeit sogar sinnvoll, die Startadresse des Unterprogramms direkt in einen Pointer zu laden und als ersten Befehl im UP einen NOP-Befehl zu schreiben.

3.7 Befehle mit dem Extension-Register

Alle Instruktionen mit dem Extensionregister sind Ein-Byte-Befehle. Sie zeichnen sich durch eine relativ kurze Ausführungszeit aus. Bei den nachfolgenden Erklärungen wird auf ausführliche Beispiele verzichtet, da die Funktion der Befehl z. B. ADD, OR usw. die Gleiche wie bei den bisher besprochenen ist.

3.7.1 Akku mit Inhalt des Extension-Registers laden LDE (load extension)

Der Inhalt des Extension-Registers wird in den Akkumulator geladen. Der alte Inhalt des Extension-Registers bleibt erhalten.

Mnemonischer Befehl: LDE
Operationskode : 40_{16}

Operation: **(EX) → (AC)**

Das Extension-Register kann als schneller Zwischenspeicher benutzt werden.

3.7.2 Akku- und Extensionregisterinhalt austauschen XAE (exchange accu and extension)

Die Inhalte von Akkumulator und Extension-Register werden ausgetauscht. Damit kann, zum Beispiel für eine serielle Ausgabe, das im Akku stehende Datenbyte ins Extension-Register gebracht werden. Gleichzeitig kann damit der beim letzten SIO-Befehl eingelesene Status des SIN-Eingangs abgefragt werden.

Mnemonischer Befehl: XAE
Operationskode : 01_{16}

Operation: **(EX) ⟷ (AC)**

3.7.3 UND-Verknüpfung mit dem Extension-Register ANE (and extension)

Der Inhalt des Akkumulators wird mit dem Inhalt des Extension-Registers UND-verknüpft. Das Ergebnis steht im Akku. Beim ANE-Befehl wird der alte Inhalt des Extension-Registers beibehalten.

Mnemonischer Befehl: ANE
Operationskode : 50_{16}
Operation: **(AC)** $\wedge$ **(EX)** $\rightarrow$ **(AC)**

3.7.4 ODER-Verknüpfung mit dem Extension-Register ORE (or extension)

Der Inhalt des Akkus wird mit dem Inhalt des Extension-Registers ODER-verknüpft. Das Resultat der Verknüpfung wird im Akkumulator abgelegt. Der Inhalt des Extension-Registers wird nicht beeinflußt.

Mnemonischer Befehl: ORE
Operationskode : 58_{16}
Operation: **(AC)** $\vee$ **(EX)** $\rightarrow$ **(AC)**

3.7.5 Exklusiv-ODER-Verknüpfung mit dem Extension-Register XRE (exclusive-or extension)

Bei dieser Instruktion werden die Inhalte von Akkumulator und Extension-Register exklusiv-ODER verknüpft und das Ergebnis im Akku abgelegt. Das Extension-Register wird nicht beeinflußt.

Mnemonischer Befehl: XRE
Operationskode : 60_{16}
Operation: **(AC)** $\veebar$ **(EX)** $\rightarrow$ **(AC)**

3.7.6 Addition mit dem Extension-Register ADE (add extension)

Hierbei wird der Inhalt des Extension-Registers zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akku und das Extension-Register behält seinen Wert. Vom Statusregister wird das CY/L- und das OV-Bit verändert, wenn die unter 3.2.6 beschriebenen Bedingungen zutreffen.

Mnemonischer Befehl: ADE
Operationskode : 70_{16}
Operation: **(AC) + (EX) + (CY/L)** $\rightarrow$ **(AC)**

3.7.7 Dezimale Addition mit dem Extension-Register DAE (decimal add extension)

Die Inhalte des Akkumulators und des Extension-Registers werden vom SCMP als zweistellige BCD-Zahlen interpretiert und entsprechend addiert. Das Ergebnis der Operation steht im Akku. Das OV-Bit wird nicht beeinflußt, für das CY/L-Bit gilt das Gleiche wie bei der bisher besprochenen Dezimaladdition.

Mnemonischer Befehl: DAE
Operationskode : 68_{16}

Operation: $(AC)_{10} + (EX)_{10} + (CY/L) \rightarrow (AC)_{10}$

3.7.8 Komplementieren und addieren mit dem Extension-Register CAE (complement and add extension)

Zum Inhalt des Akkumulators wird das Einerkomplement des Extension-Registers und der Wert des CY/L-Bit addiert. Das Resultat steht im Akku. Im Statusregister werden das OV- und das CY/L-Bit beeinflußt. Die Bedingungen sind in Kapitel 3.2.8 zu finden.

Mnemonischer Befehl: CAE
Operationskode : 78_{16}

Operation: $(AC) + (\overline{EX}) + (CY/L) \rightarrow (AC)$

3.8 Serielle Ein-/Ausgabe, Schiebe- und Rotationsbefehle

Bei allen Instruktionen dieser Gruppe handelt es sich um Ein-Byte-Befehle. Weiterhin sind es durchweg sehr »schnelle« Befehle.

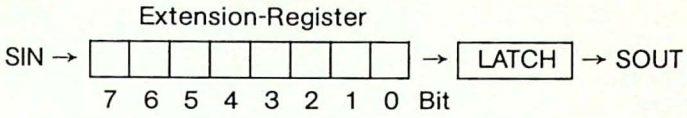
3.8.1 Serieller Ein-/Ausgabe-Befehl SIO (serial input/output)

Diese Anweisung bewirkt ein Rechtsschieben des Extension-Register-Inhalts um ein Bit (eine Stelle), wobei das niederwertigste Bit, über ein Latch zwischengespeichert, am SOUT-Ausgang des SCMP erscheint. Es bleibt dort solange anstehen bis ein erneuter SIO-Befehl ausgeführt wird.
Zur gleichen Zeit, wie das niederwertigste Bit an den Ausgang SOUT geschoben wird, wird die am Eingang SIN anstehende Information (»0« oder »1«) in das höchstwertige Bit des Extension-Registers transportiert. Durch achtmaliges Anwenden dieser Instruktion kann ein Byte seriell eingelesen oder ausgegeben werden. Es ist sogar denkbar, daß Ein- und Ausgabe gleichzeitig ablaufen.

Mnemonischer Befehl: SIO
Operationskode : 19_{16}

Operation: **(EX(I)) → (EX(I−1))**
SIN → (EX(7))
(EX(0)) → SOUT

Zur Verdeutlichung der Wirkungsweise dieses Befehls soll nachfolgendes Bild dienen.



3.8.2 Rechtsschieben des Akkuinhaltes SR (shift right)

Bei der Ausführung dieser Anweisung wird der Akkumulatorinhalt eine Stelle nach rechts geschoben. Die höchstwertige Stelle des Akkus wird mit »0« aufgefüllt, während der alte Wert des niederwertigsten Bits verloren geht.

Mnemonischer Befehl: SR
Operationskode : $1C_{16}$

Operation: **(AC(I)) → (AC(I−1))**
0 → (AC(7))

Hat der Akkumulator vor der Ausführung des SR-Befehls den Inhalt $3A_{16}$, so hat er danach den Inhalt $1D_{16}$.
Doch was hat der Befehl noch bewirkt?
Der Inhalt des Akkus wurde durch zwei dividiert. Wendet man den Befehl noch einmal oder mehrmals an, so wird der ursprüngliche Inhalt durch zwei, vier, acht usw. geteilt. Nachkommastellen gehen verloren, wenn man sie nicht programmtechnisch sichert.

3.8.3 Rechtsschieben mit CY/L des Akkuinhaltes SRL (shift right with link)

Im Prinzip wird dieser Befehl ebenso wie der SR-Befehl ausgeführt. Bei der SRL-Instruktion wird jedoch nicht »0«, sondern der Wert des CY/L-Bits in die höchstwertige Stelle des Akkus geschoben. Das CY/L-Bit wird nicht beeinflußt.

Mnemonischer Befehl: SRL
Operationskode : $1D_{16}$

Operation: **(AC(I)) → (AC(I−1))**
(CY/L) → (AC(7))

3.8.4 Akkuinhalt nach rechts rotieren RR (rotate right)

Der Inhalt des Akkumulators wird eine Stelle nach rechts geschoben, wobei das niederwertigste Bit ins höchstwertige Bit geschoben wird.

Mnemonischer Befehl: RR
Operationskode : $1E_{16}$

Operation: **(AC(I)) → (AC(I−1))**
(AC(0)) → (AC(7))

3.8.5 Akkuinhalt rechts rotieren mit CY/L RRL (rotate right with link)

Bei diesem Befehl wird ebenfalls der Akkuinhalt um eine Stelle nach rechts geschoben. Das niederwertigste Bit wird jedoch ins CY/L-Bit transportiert und der ursprüngliche Wert des CY/L-Bits erscheint im höchstwertigen Bit des Akkumulators.

Mnemonischer Befehl: RRL
Operationskode : $1F_{16}$

Operation: **(AC(I)) → (AC(I−1))**
(AC(0)) → (CY/L)
(CY/L) → (AC(7))

Es ist ein wenig schade, daß unser SCMP keine Linksschiebe-Befehle in seinem Befehlssatz hat. Aber auch hier gibt es einen Ersatz. Addiert man den Inhalt des Akkus auf sich selbst, so entspricht dies einer Multiplikation mit zwei oder einem Schieben nach links.
Dazu ein kleines Beispiel. Das CY/L-Bit sei »0«.

```
LDI 05      ;  05₁₆ → (AC)
XAE         ;  (AC) → (EX)
LDE         ;  (EX) → (AC)
ADE         ;  05₁₆ + 05₁₆ = 0A₁₆ → (AC), CY/L bleibt »0«
```

Ein wenig umständlich, aber es funktioniert.

3.9 Der Verzögerungsbefehl DLY (delay)

Dieser Zwei-Byte-Befehl ist eine Besonderheit unseres SCMP die sonst kein anderer Mikroprozessor zu bieten hat.
Mit dieser Anweisung lassen sich Verzögerungen realisieren, die bei einer Taktfrequenz von 2 MHz, zwischen 26 Mikrosekunden und 263 Millisekunden liegen.
Das zweite Byte des Befehls gibt die groben Grenzen der Verzögerungszeit an, die

Feineinstellung wird über den Akkumulatorinhalt vorgenommen. Nach Abarbeiten des DLY-Befehls hat der Akku den Inhalt FF_{16}.

Mnemonischer Befehl: DLY XX
Operationskode : 8F XX

Der DLY-Befehl kann nicht durch einen Interrupt unterbrochen werden.

Für die Verzögerungszeit gilt:

Verzögerungszeit $T_V = (2 \cdot (AC)_{10} + 514_{10} \cdot XX_{10} + 13_{10})$ in MZ

Die Zeit T_V ist in Mikrozyklen »MZ« angegeben, weil die Dauer eines Mikrozyklus abhängig von der Taktfrequenz ist. Hier ist erstmals der Begriff »Mikrozyklus« aufgetaucht. Was es damit für eine Bewandnis hat wollen wir uns gleich anschauen.
Wie wir bereits in Kapitel 2 gesehen haben laufen bei der Ausführung der Instruktionen in der CPU verschiedene Mikroprogramme ab. Es treten dabei Lese- und Schreibzyklen auf. Der Gesamtablauf wird von dem am Mikroprozessor angelegten Systemtakt gesteuert. Der SCMP II führt nun in der vierfachen Periodendauer des Systemtakts genau einen Mikrozyklus aus. Die Befehlsausführung benötigt wie schon gesagt mehrere (5 bis 22) Mikrozyklen. Aus der Anzahl der Mikrozyklen und der Taktfrequenz kann die Befehlsausführungszeit berechnet werden. Die Dauer eines MZ ist 4/Taktfrequenz.
In den Befehlstabellen in den Kapiteln 3.11 und 3.12 sind zu den einzelnen Befehlen die Anzahl der MZ angegeben. Für 2 MHz sind außerdem schon die Befehlsausführungszeiten aufgeführt. Bei 4 MHz Takt entsprechen die Anzahl der Mikrozyklen der Ausführungszeit in Mikrosekunden. Doch zurück zu unserem DLY-Befehl.
Für die Verzögerungszeit T_V haben wir bereits eine Berechnungsformel angegeben. In der Praxis ist jedoch meist die Zeit T_V vorgegeben. Unbekannt sind der Wert des Displacements und der Inhalt des Akkumulators. Durch entsprechendes Umstellen der Ausgangsgleichung erhält man für das Displacement XX und den Akku-Inhalt je eine Gleichung. Sie sind nachfolgend aufgeführt.

1) $\mathbf{(AC)_{10} = ((T_V/MZ - 13)/2) - 257_{10} \cdot XX_{10}}$

2) $\mathbf{XX_{10} = (T_V/MZ - 13)/514}$

XX = Displacement
T_V = Verzögerungszeit in Mikrozyklen

Beispiel: Es soll eine Verzögerungszeit von genau 1 ms bei einem Systemtakt von 2 MHz realisiert werden.

1 MZ dauert $1/2 \cdot 10^{-6} \cdot 4 = 2$ Mikrosekunden
1 ms = 1000 µs ergibt 500 MZ

$XX_{10} = (500 - 13)/514 = 0{,}94 = 00_{10}$ (nur ganzzahligen Anteil)
$(AC)_{10} = (500 - 13)/2 - 257 \cdot 0 = 243{,}5$ hier kann 243_{10} oder 244_{10} benutzt werden. In beiden Fällen kommt man nicht genau auf 1 ms.

Wir wählen $243_{10} = F3_{16}$ als Akkumulatorinhalt.

Unsere Anweisungen an den SCMP müßten lauten:
LDI F3
DLY 00

Setzt man die errechneten Werte in die Gleichung für T_V ein, so erhält man eine Zeit von 0,998 ms.

Die Ungenauigkeit kommt daher, daß wir nur ganze Zahlen als Displacement bzw. Akkuinhalt einsetzen können. Will man sehr genaue Verzögerungszeiten einstellen, so muß man etwas mit Akkuinhalt und Displacement »probieren«.

3.10 Verschiedene Ein-Byte-Befehle

Hier sind die Instruktionen eingeordnet, die in keine der anderen Gruppen passen. Es sind Anweisungen die ein Byte lang sind.

3.10.1 Leerbefehl NOP (no operation)

Alle nicht definierten Operationskodes interpretiert der SCMP als NOP-Befehl. Abhängig vom Wert der 16^1-Stelle des Kodes wird ein Ein-Byte-NOP oder ein Zwei-Byte-NOP ausgeführt. Ist die höherwertige Stelle des nicht definierten Kodes gleich oder größer 8_{16}, so führt der SCMP einen Zwei-Byte-NOP aus. Im anderen Fall einen Ein-Byte-NOP. Doch wir wollen uns nur die zulässige NOP-Instruktion anschauen.

Mnemonischer Befehl: NOP
Operationskode : 08_{16}

Operation: **(PC) + 1 → (PC)**

Mit jedem NOP wird der Programm Counter um eins erhöht. Der Befehl läßt sich somit sehr gut als »Platzhalter« einsetzen.

3.10.2 Die Instruktion HALT (halt)

Bei der Ausführung dieses Befehls wird das H-Flag auf »1« gesetzt und während der NADS-Zeit am Datenbit 7 des Mikroprozessors ausgegeben. Es kann, bei entsprechender äußerer Beschaltung, zum programmierten Halten des Prozessors benutzt werden.

Mnemonischer Befehl: HALT
Operationskode : 00_{16}

Operation: **(PC) + 1 → (PC)**
»1« → H-Flag

3.10.3 Interrupt sperren DINT (disable interrupt)

Mit diesem Befehl wird das IE-Bit (Interrupt Enable) im Statusregister auf »0« gesetzt. Damit wirkt der Eingang SENSE A ebenso wie der SENSE B-Eingang als einfacher Eingang. Auf Signale an SENSE A erfolgt keine Reaktion des Prozessors.

Mnemonischer Befehl: DINT
Operationskode : 04_{16}

Operation: »0« → **IE-Bit**

3.10.4 Interrupt erlauben IEN (interrupt enable)

Die Instruktion IEN setzt das IE-Bit des Statusregisters auf »1«. Eine Interruptreaktion des SCMP kann damit stattfinden. Diese wird durch ein »1«-Signal an SENSE A eingeleitet.

Mnemonischer Befehl: IEN
Operationskode : 05_{16}

Operation: »1« → **IE-Bit**

3.10.5 Carry/Link-Bit (CY/L) löschen CCL (clear carry/link)

Mit diesem Befehl wird das CY/L-Bit im Statusregister auf »0« gesetzt.

Mnemonischer Befehl: CCL
Operationskode : 02_{16}

Operation: »0« → **CY/L-Bit**

3.10.6 Carry/Link-Bit (CY/L) setzen SCL (set carry/link)

Das CY/L-Bit wird mit dieser Anweisung auf »1« gesetzt.

Mnemonischer Befehl: SCL
Operationskode : 03_{16}

Operation: »1« → **CY/L-Bit**

3.10.7 Akkuinhalt in das Statusregister schreiben CAS (copy accu in status)

Der Befehl CAS bewirkt, daß der Inhalt des Akkumulators ins Statusregister kopiert wird. Ausgenommen davon sind die beiden SENSE-Eingänge. Der alte Inhalt des Akkus geht dabei nicht verloren.

Mnemonischer Befehl: CAS
Operationskode : 07_{16}

Operation: **(AC) → (ST)**

3.10.8 Statusregisterinhalt in den Akku kopieren CSA (copy status in accu)

Die umgekehrte Funktion wie beim CAS-Befehl wird beim CSA-Befehl ausgeführt. Der Inhalt des Statusregisters einschließlich der beiden SENSE-Eingänge wird in den Akkumulator kopiert. Das Statusregister wird dabei nicht verändert.

Mnemonischer Befehl: CSA
Operationskode : 06_{16}

Operation: **(ST) → (AC)**

3.11 Befehlsliste geordnet nach dem sedezimalen Kode

OP-Code	Mnem. Bef.	MZ	Zeit	Funktion	Status
00	HALT	8	16	setzt H-Flag; (PC) + 1 → (PC)	
01	XAE	7	14	(AC) ↔ (EX)	
02	CCL	5	10	»0« → (CY/L)	CY/L
03	SCL	5	10	»1« → (CY/L)	CY/L
04	DINT	6	12	»0« → (IE)	IE
05	IEN	6	12	»1« → (IE)	IE
06	CSA	5	10	(ST) → (AC), einschl. SENSE A, B	
07	CAS	6	12	(AC) → (ST), ohne SENSE A, B	alle
08	NOP	5	10	Leerbefehl, (PC) + 1 → (PC)	
19	SIO	5	10	(EX(I)) → (EX(I−1)), SIN → (EX(7)), (EX(0)) → SOUT	
1C	SR	5	10	(AC(I)) → (AC(I−1)), »0« → (AC(7))	
1D	SRL	5	10	(AC(I)) → (AC(I−1)), (CY/L) → (AC(7))	
1E	RR	5	10	(AC(I)) → (AC(I−1)), (AC(0)) → (AC(7))	
1F	RRL	5	10	(AC(I)) → (AC(I−1)), (CY/L) → (AC(7)), (AC(0)) → (CY/L)	
30	XPAL 0	8	16	(AC) ↔ (LOB(PC))	
31	XPAL 1	8	16	(AC) ↔ (LOB(P1))	

OP-Code	Mnem. Bef.	MZ	Zeit	Funktion	Status
32	XPAL 2	8	16	(AC) ↔ (LOB(P2))	
33	XPAL 3	8	16	(AC) ↔ (LOB(P3))	
34	XPAH 0	8	16	(AC) ↔ (HOB(PC))	
35	XPAH 1	8	16	(AC) ↔ (HOB(P1))	
36	XPAH 2	8	16	(AC) ↔ (HOB(P2))	
37	XPAH 3	8	16	(AC) ↔ (HOB(P3))	
3C	XPPC 0	7	14	(PC) ↔ (PC)	
3D	XPPC 1	7	14	(PC) ↔ (P1)	
3E	XPPC 2	7	14	(PC) ↔ (P2)	
3F	XPPC 3	7	14	(PC) ↔ (P3)	
40	LDE	6	12	(EX) → (AC)	
50	ANE	6	12	(AC) $\wedge$ (EX) → (AC)	
58	ORE	6	12	(AC) $\vee$ (EX) → (AC)	
60	XRE	6	12	(AC) $\veebar$ (EX) → (AC)	
68	DAE	11	22	$(AC)_{10} + (EX)_{10} + (CY/L) \rightarrow (AC)_{10}$	CY/L
70	ADE	7	14	$(AC)_{16} + (EX)_{16} + (CY/L) \rightarrow (AC)_{16}$	CY/L, OV
78	CAE	8	16	$(AC)_{16} + (\overline{EX})_{16} + (CY/L) \rightarrow (AC)_{16}$	CY/L, OV
8F XX	DLY XX	*	*	13*−131593*MZ, 26*−263186* µs; $T_V = (2.(AC)_{10} + 514_{10} \cdot XX_{10} + 13_{10})$ in MZ	
90 XX	JMP XX	11	22	EA → (PC), EA relativ zu PC	
91 XX	JMP XX(1)	11	22	EA → (PC), EA relativ zu P1	
92 XX	JMP XX(2)	11	22	EA → (PC), EA relativ zu P2	
93 XX	JMP XX(3)	11	22	EA → (PC), EA relativ zu P3	
94 XX	JP XX	11	22	EA → (PC), wenn AC (7) = 0; bei nein 9MZ	
95 XX	JP XX(1)	11	22	EA → (PC), wenn AC (7) = 0; bei nein 9MZ	
96 XX	JP XX(2)	11	22	EA → (PC), wenn AC (7) = 0; bei nein 9MZ	
97 XX	JP XX(3)	11	22	EA → (PC), wenn AC (7) = 0; bei nein 9MZ	
98 XX	JZ XX	11	22	EA → (PC), wenn (AC) = 0; bei nein 9MZ	
99 XX	JZ XX(1)	11	22	EA → (PC), wenn (AC) = 0; bei nein 9MZ	
9A XX	JZ XX(2)	11	22	EA → (PC), wenn (AC) = 0; bei nein 9MZ	
9B XX	JZ XX(3)	11	22	EA → (PC), wenn (AC) = 0; bei nein 9MZ	

OP-Code	Mnem. Bef.	MZ	Zeit	Funktion	Status
9C XX	JNZ XX	11	22	EA → (PC), wenn (AC) = 0; bei nein 9MZ	
9D XX	JNZ XX(1)	11	22	EA → (PC), wenn (AC) = 0; bei nein 9MZ	
9E XX	JNZ XX(2)	11	22	EA → (PC), wenn (AC) = 0; bei nein 9 MZ	
9F XX	JNZ XX(3)	11	22	EA → (PC), wenn (AC) = 0; bei nein 9MZ	
A8 XX	ILD XX	22	44	(EA) + 1 → (EA), (AC); relativ zu PC	
A9 XX	ILD XX(1)	22	44	(EA) + 1 → (EA), (AC); relativ zu P1	
AA XX	ILD XX(2)	22	44	(EA) + 1 → (EA), (AC); relativ zu P2	
AB XX	ILD XX(3)	22	44	(EA) + 1 → (EA), (AC); relativ zu P3	
B8 XX	DLD XX	22	44	(EA) − 1 → (EA), (AC); relativ zu PC	
B9 XX	DLD XX(1)	22	44	(EA) − 1 → (EA), (AC); relativ zu P1	
BA XX	DLD XX(2)	22	44	(EA) − 1 → (EA), (AC); relativ zu P2	
BB XX	DLD XX(3)	22	44	(EA) − 1 → (EA), (AC); relativ zu P3	
C0 XX	LD XX	18	36	(EA) → (AC); relativ zu PC	
C1 XX	LD XX(1)	18	36	(EA) → (AC); relativ zu P1	
C2 XX	LD XX(2)	18	36	(EA) → (AC); relativ zu P2	
C3 XX	LD XX(3)	18	36	(EA) → (AC); relativ zu P3	
C4 XX	LDI XX	10	20	XX → (AC)	
C5 XX	LD @XX(1)	18	36	(P1) − XX → (P1), (EA) → (AC), (P1) + XX → (P1)	
C6 XX	LD @XX(2)	18	36	(P2) − XX → (P2), (EA) → (AC), (P2) + XX → (P2)	
C7 XX	LD @XX(3)	18	36	(P3) − XX → (P3), (EA) → (AC), (P3) + XX → (P3)	
C8 XX	ST XX	18	36	(AC) → (EA); relativ zu PC	
C9 XX	ST XX(1)	18	36	(AC) → (EA); relativ zu P1	
CA XX	ST XX(2)	18	36	(AC) → (EA); relativ zu P2	
CB XX	ST XX(3)	18	36	(AC) → (EA); relativ zu P3	
CD XX	ST @XX(1)	18	36	(P1) − XX → (P1), (AC) → (EA), (P1) + 1 → (P1)	
CE XX	ST @XX(2)	18	36	(P2) − XX → (P2), (AC) → (EA), (P2) + 1 → (P2)	
CF XX	ST @XX(3)	18	36	(P3) − XX → (P3), (AC) → (EA), (P3) + 1 → (P3)	
D0 XX	AND XX	18	36	(AC) ∧ (EA) → (AC); relativ zu PC	
D1 XX	AND XX(1)	18	36	(AC) ∧ (EA) → (AC); relativ zu P1	
D2 XX	AND XX(2)	18	36	(AC) ∧ (EA) → (AC); relativ zu P2	
D3 XX	AND XX(3)	18	36	(AC) ∧ (EA) → (AC); relativ zu P3	

OP-Code	Mnem. Bef.	MZ	Zeit	Funktion	Status
D4 XX	ANI XX	10	20	(AC) $\wedge$ XX $\rightarrow$ (AC)	
D5 XX	AND@XX(1)	18	36	(P1) − XX $\rightarrow$ (P1), (AC) $\wedge$ (EA) $\rightarrow$ (AC), (P1) + XX $\rightarrow$ (P1)	
D6 XX	AND@XX(2)	18	36	(P2) − XX $\rightarrow$ (P2), (AC) $\wedge$ (EA) $\rightarrow$ (AC), (P2) + XX $\rightarrow$ (P2)	
D7 XX	AND@XX(3)	18	36	(P3) − XX $\rightarrow$ (P3), (AC) $\wedge$ (EA) $\rightarrow$ (AC), (P3) + XX $\rightarrow$ (P3)	
D8 XX	OR XX	18	36	(AC) $\vee$ (EA) $\rightarrow$ (AC); relativ zu PC	
D9 XX	OR XX(1)	18	36	(AC) $\vee$ (EA) $\rightarrow$ (AC); relativ zu P1	
DA XX	OR XX(2)	18	36	(AC) $\vee$ (EA) $\rightarrow$ (AC); relativ zu P2	
DB XX	OR XX(3)	18	36	(AC) $\vee$ (EA) $\rightarrow$ (AC); relativ zu P3	
DC XX	ORI XX	10	20	(AC) $\vee$ XX $\rightarrow$ (AC)	
DD XX	OR @XX(1)	18	36	(P1) − XX $\rightarrow$ (P1), (AC) $\vee$ (EA) $\rightarrow$ (AC), (P1) + XX $\rightarrow$ (P1)	
DE XX	OR @XX(2)	18	36	(P2) − XX $\rightarrow$ (P2), (AC) $\vee$ (EA) $\rightarrow$ (AC), (P2) + XX $\rightarrow$ (P2)	
DF XX	OR @XX(3)	18	36	(P3) − XX $\rightarrow$ (P3), (AC) $\vee$ (EA) $\rightarrow$ (AC), (P3) + XX $\rightarrow$ (P3)	
E0 XX	XOR XX	18	36	(AC) $\veebar$ (EA) $\rightarrow$ (AC); relativ zu PC	
E1 XX	XOR XX(1)	18	36	(AC) $\veebar$ (EA) $\rightarrow$ (AC); relativ zu P1	
E2 XX	XOR XX(2)	18	36	(AC) $\veebar$ (EA) $\rightarrow$ (AC); relativ zu P2	
E3 XX	XOR XX(3)	18	36	(AC) $\veebar$ (EA) $\rightarrow$ (AC); relativ zu P3	
E4 XX	XRI XX	10	20	(AC) $\veebar$ XX $\rightarrow$ (AC)	
E5 XX	XOR@XX(1)	18	36	(P1) − XX $\rightarrow$ (P1), (AC) $\veebar$ (EA) $\rightarrow$ (AC), (P1) + XX $\rightarrow$ (P1)	
E6 XX	XOR@XX(2)	18	36	(P2) − XX $\rightarrow$ (P2), (AC) $\veebar$ (EA) $\rightarrow$ (AC), (P2) + XX $\rightarrow$ (P2)	
E7 XX	XOR@XX(3)	18	36	(P3) − XX $\rightarrow$ (P3), (AC) $\veebar$ (EA) $\rightarrow$ (AC), (P3) + XX $\rightarrow$ (P3)	
E8 XX	DAD XX	23	46	$(AC)_{10}$ + $(EA)_{10}$ + (CY/L) $\rightarrow$ (AC); rel. PC; CY/L	
E9 XX	DAD XX(1)	23	46	$(AC)_{10}$ + $(EA)_{10}$ + (CY/L) $\rightarrow$ (AC); rel. P1; CY/L	
EA XX	DAD XX(2)	23	46	$(AC)_{10}$ + $(EA)_{10}$ + (CY/L) $\rightarrow$ (AC); rel. P2;	CY/L
EB XX	DAD XX(3)	23	46	$(AC)_{10}$ + $(EA)_{10}$ + (CY/L) $\rightarrow$ (AC); rel. P3;	CY/L
EC XX	DAI XX	15	30	$(AC)_{10}$ + XX_{10}(CY/L) $\rightarrow$ $(AC)_{10}$;	CY/L
ED XX	DAD@XX(1)	23	46	OP. wie DAD zus. Pointer 1 INC/DEC	CY/L
EE XX	DAD@XX(2)	23	46	OP. wie DAD zus. Pointer 2 INC/DEC	CY/L
EF XX	DAD@XX(3)	23	46	OP. wie DAD zus. Pointer 3 INC/DEC	CY/L

OP-Code	Mnem. Bef.	MZ	Zeit	Funktion	Status
F0 XX	ADD XX	19	38	$(AC)_{16} + (EA)_{16} + (CY/L) \rightarrow (AC)$; rel. PC;	CY/L, OV
F1 XX	ADD XX(1)	19	38	$(AC)_{16} + (EA)_{16} + (CY/L) \rightarrow (AC)$; rel. P1;	CY/L, OV
F2 XX	ADD XX(2)	19	38	$(AC)_{16} + (EA)_{16} + (CY/L) \rightarrow (AC)$; rel. P2;	CY/L, OV
F3 XX	ADD XX(3)	19	38	$(AC)_{16} + (EA)_{16} + (CY/L) \rightarrow (AC)$; rel. P3;	CY/L, OV
F4 XX	ADI XX	11	22	$(AC)_{16} + XX_{16} + (CY/L) \rightarrow (AC)_{16}$;	CY/L, OV
F5 XX	ADD@XX(1)	19	38	OP. wie ADD zus. Pointer 1 INC/DEC;	CY/L, OV
F6 XX	ADD@XX(2)	19	38	OP. wie ADD zus. Pointer 2 INC/DEC;	CY/L, OV
F7 XX	ADD@XX(3)	19	38	OP. wie ADD zus. Pointer 3 INC/DEC;	CY/L, OV
F8 XX	CAD XX	20	40	$(AC)_{16} + (\overline{EA})_{16} + (CY/L) \rightarrow (AC)$; rel. PC;	CY/L, OV
F9 XX	CAD XX(1)	20	40	$(AC)_{16} + (\overline{EA})_{16} + (CY/L) \rightarrow (AC)$; rel. P1;	CY/L, OV
FA XX	CAD XX(2)	20	40	$(AC)_{16} + (\overline{EA})_{16} + (CY/L) \rightarrow (AC)$; rel. P2;	CY/L, OV
FB XX	CAD XX(3)	20	40	$(AC)_{16} + (\overline{EA})_{16} + (CY/L) \rightarrow (AC)$; rel. P3;	CY/L, OV
FC XX	CAI XX	12	24	$(AC)_{16} + \overline{XX}_{16} + (CY/L) \rightarrow (AC)_{16}$;	CY/L, OV
FD XX	CAD@XX(1)	20	40	OP. wie CAD zus. Pointer 1 INC/DEC;	CY/L, OV
FE XX	CAD@XX(2)	20	40	OP. wie CAD zus. Pointer 2 INC/DEC;	CY/L, OV
FF XX	CAD@XX(3)	20	40	OP. wie CAD zus. Pointer 3 INC/DEC;	CY/L, OV

@ steht für auto-inkrement/dekrement.
Das Zeichen @ wird auch als »Klammeraffe« bezeichnet.
$\wedge$ ist das Symbol für die logische UND-Verknüpfung.
$\vee$ ist das Symbol für die logische ODER-Verknüpfung.
$\veebar$ ist das Symbol für die logische Exklusiv-ODER-Verknüpfung.
$\overline{XX}$ oder $(\overline{EA})$ bedeutet Einerkomplement von XX bzw. Einerkomplement vom Inhalt der effektiven Adresse »EA«.

3.12 Befehlsliste der Basisbefehle in alphabetischer Folge

Mnemonic	Bytes	Basis OP-Code	engl. Funktionsbeschreibung
ADD	2	F0	**ADD** memory to accu
ADE	1	70	**AD**d **E**xtension to accu
ADI	2	F4	**AD**d **I**mmediate to accu
AND	2	D0	**AND** accu with memory
ANE	1	50	**AN**d accu with **E**xtension register
ANI	2	D4	**AN**d accu with **I**mmediate following data
CAD	2	F8	**C**omplement and **AD**d memory to accu
CAE	1	78	**C**omplement and **A**dd **E**xtension register to accu
CAI	2	FC	**C**omplement and **A**dd **I**mmediate following data to accu
CAS	1	07	**C**opy **A**ccu to **S**tatus register
CCL	1	02	**C**lear **C**arry/**L**ink bit in status register
CSA	1	06	**C**opy **S**tatus register to **A**ccu
DAD	2	E8	**D**ecimal **AD**d memory to accu
DAE	1	68	**D**ecimal **A**dd **E**xtension register to accu
DAI	2	EC	**D**ecimal **A**dd **I**mmediate following data to accu
DINT	1	04	**D**isable **INT**errupts
DLD	2	B8	**D**ecrement and **LoaD** memory into accu
DLY	2	8F	**DeLaY**
HALT	1	00	**HALT** (sets H-flag)
IEN	1	05	**I**nterrupt **EN**able
ILD	2	A8	**I**ncrement and **LoaD** memory into accu
JMP	2	90	**JuMP** allways
JNZ	2	9C	**J**ump if accu **N**ot **Z**ero
JP	2	94	**J**ump if accu **P**ositive
JZ	2	98	**J**ump if accu **Z**ero
LD	2	C0	**LoaD** accu from memory
LDE	1	40	**LoaD** accu from **E**xtension register
LDI	2	C4	**LoaD** **I**mmediate following data in accu
NOP	1	08	**N**o **OP**eration
OR	2	D8	**OR** accu with memory
ORE	1	58	**OR** accu with **E**xtension register
ORI	2	DC	**OR** accu with **I**mmediate following data
RR	1	1E	**R**otate **R**ight accu by one bit
RRL	1	1F	**R**otate **R**ight accu by one bit with carry/**L**ink
SCL	1	03	**S**et **C**arry/**L**ink bit
SIO	1	19	**S**erial **I**nput/**O**utput
SR	1	1C	**S**hift **R**ight accu by one bit

Mnemonic	Bytes	Basis OP-Code	engl. Funktionsbeschreibung
SRL	1	1D	**S**hift **R**ight by one bit with carry/**L**ink
ST	2	C8	**ST**ore accu to memory
XAE	1	01	e**X**change **A**ccu and **E**xtension register
XOR	2	E0	e**X**clusive **OR** accu with memory
XPAH	1	34	e**X**change **P**ointer with **A**ccu (**H**igh)
XPAL	1	30	e**X**change **P**ointer with **A**ccu (**L**ow)
XPPC	1	3C	e**X**change **P**ointer with **P**rogramm **C**ounter
XRE	1	60	e**X**clusive o**R** accu with **E**xtension register
XRI	2	E4	e**X**clusive o**R** accu with **I**mmediate following data

4. Anwendungs- und Programmierbeispiele

In diesem Kapitel soll anhand verschiedener Programme in die Programmierungstechnik bei Mikroprozessoren eingeführt werden. Spezielle Abläufe werden zusätzlich nach dem »warum« durchleuchtet. Da abhängig von einem eventuell vorhandenen Entwicklungssystem die Aus- und Eingabe von Daten unterschiedlich gelöst sein kann, sind die Programme so weit wie möglich »relocatable« (im Speicher frei verschiebbar) geschrieben. Bei Ausnahmen sind die zu ändernden Adressen bzw. Pointerinhalte gesondert aufgeführt.
Alle Programme sind außerdem nicht mit sogenannten »Programmiertricks« realisiert, sondern sind bewußt einfach strukturiert worden, um dem Neuling das Verständnis zu erleichtern.
Im ersten Programm wird auf alle Einzelheiten ausführlich eingegangen. Damit soll deutlich werden, daß zwischen Idee und fertigem, lauffähigen Programm eine Menge Arbeit liegt.

4.1 Das Programm »STOPPUHR«

Es soll eine »Software-Stoppuhr« realisiert werden, mit deren Hilfe Zeiten zwischen 0,01 Sekunden und 99 Minuten mit einer Auflösung von einer hundertstel Sekunde gemessen werden können.
Als Anzeigemedium wird eine fiktive 6-stellige Siebensegmentanzeige verwendet. Fiktiv deshalb, weil die Anzeige nur im Programm als Adresse (Speicherstelle) vorhanden ist. Selbstverständlich kann bei Bedarf und schaltungstechnischer Möglichkeit eine Anzeige auf diese Art und Weise betrieben werden.
Bevor man daran geht ein Programm zu schreiben, wird auch als kodieren bezeichnet, muß man sich zunächst über die Randbedingungen klar werden. Diese sind in unserem Fall:

a) Wie soll die Anzeige aussehen?
b) Wie sollen Start und Stop ausgelöst werden?
c) Wie soll die Anzeige wieder gelöscht werden?
d) Welche Funktionen bzw. Funktionsblöcke werden benötigt?

Ist der Rahmen somit abgesteckt, kann man daran gehen den Programmablauf in Form eines Flußdiagramms bzw. Programmablaufplanes zu entwerfen.

Zu Punkt a)
Eine Siebensegmentanzeige hat sieben Anschlüsse für die Segmente, einen Anschluß für den Dezimalpunkt und einen oder mehrere Pins für die gemeinsamen Anoden bzw. Kathoden; je nach Ausführung.
Mit einem Byte können wir demnach eine solche Anzeige ansteuern. Je Segment, einschließlich des Dezimalpunktes, steht uns ein Bit zur Verfügung. Jede Stelle bzw. jedes Byte wird über eine eigene Adresse angesprochen. Der Zusammenhang zwischen Adresse und Anzeigestelle und zwischen Bitposition und Segment bzw. Dezimalpunkt geht aus Bild 4.1.1 hervor.
Die Segmente sind mit den Buchstaben A bis G bezeichnet, der Dezimalpunkt mit DP. Aus der Zuordnung der Bitposition zu den Segmenten ergibt sich folgende Tabelle. In ihr sind den Ziffern 0 bis 9 und dem Dezimalpunkt je ein entsprechender sedezimaler Kode zugewiesen.
Im Programm hat der Anfang der Tabelle den symbolischen Namen DISTB mit der absoluten Adresse $05E0_{16}$.

Adresse	Anzeigeziffer	sedezimaler Kode
$5E0_{16}$	0	$3F_{16}$
$5E1_{16}$	1	06_{16}
$5E2_{16}$	2	$5B_{16}$
$5E3_{16}$	3	$4F_{16}$
$5E4_{16}$	4	66_{16}
$5E5_{16}$	5	$6D_{16}$
$5E6_{16}$	6	$7D_{16}$
$5E7_{16}$	7	07_{16}
$5E8_{16}$	8	$7F_{16}$
$5E9_{16}$	9	$6F_{16}$
$5EA_{16}$	DP	80_{16}

Nach Programmstart sollen zunächst alle Stellen des Displays »Null« anzeigen. Während die »Stoppuhr« läuft, erscheint auf der Anzeige immer die aktuelle sich ändernde Zeit, die nach dem Stoppen der Uhr quasi »eingefroren« wird.

Zu Punkt b)
Der Start der Zeitmessung erfolgt mit einem »1«-Signal am Eingang SENSE A. Angehalten wird die Stoppuhr mit einem »1«-Signal am SENSE B Eingang. Die aktuelle Zeit bleibt erhalten, bis sie wieder gelöscht wird.

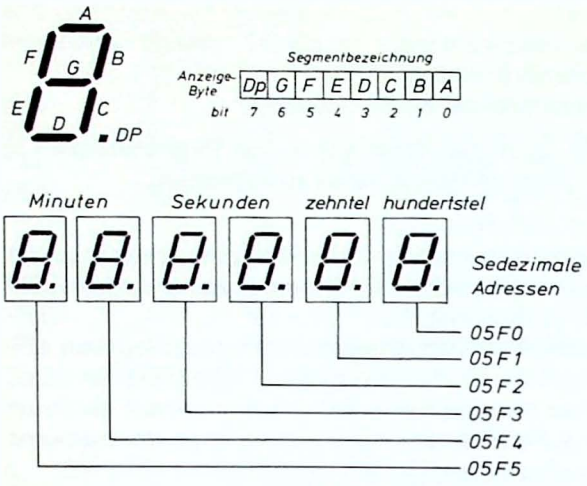


Bild 4.1.1:
Zuordnung von Bitpositionen zu Segment und Speicheradresse zu Anzeigestelle

Zu Punkt c)
Werden nach dem Stop der Uhr beide Eingänge SENSE A und SENSE B auf »1« gelegt, so wird die Anzeige gelöscht. Es wird die Zeit 00.00.00 angezeigt. Das Stoppuhrprogramm wartet danach in einer Schleife auf einen neuen Start.

Zu Punkt d)

An Funktionsblöcken, auch Programm-Moduln genannt, werden benötigt:

1) Initialisierungsmodul; in ihm wird die Anzeige auf Null gesetzt und die Pointer und Zähler erhalten ihre Basiswerte.
2) Im Start-Warteblock wird in einer Schleife dauernd der Eingang SENSE A abgefragt.
3) Der Zählmodul addiert die hundertstel und zehntel Sekunden und übergibt das Ergebnis an den Anzeigemodul.
4) Im Anzeigemodul wird aus der binären Zeitdarstellung die Siebensegmentdarstellung erzeugt und die Displays entsprechend angesteuert.
5) Nach betätigtem Stop wird die Anzeige aktualisiert und danach in einer Schleife auf die Reset-Eingabe (SENSE A und B gleich »1«) gewartet. Ist die Reset-Bedingung erfüllt, werden alle Anzeigestellen auf Null gebracht. Danach wird wieder der Start-Warteblock ausgeführt.

Nachdem das Flußdiagramm erstellt ist, fertigt man sich sinnvollerweise eine Variablenliste an, in der alle zu diesem Zeitpunkt bereits bekannten Variablen oder Konstanten aufgeführt sind. Auch sollte an dieser Stelle bereits berücksichtigt werden ob das fertige Programm später einmal auch in einem PROM lauffähig sein soll oder ob es nur im RAM-Bereich eingesetzt wird. Wie sicher bekannt ist, können in einem ROM, PROM oder EPROM nur Daten gelesen werden. Dieser Aspekt muß bei der Festlegung speziell der Variablen berücksichtigt werden.

Bevor wir nun für unser Beispiel die Variablenliste erstellen wollen, noch folgendes. Auch wenn die ganze Vorarbeit etwas zu aufwendig erscheint, sie ist es nicht. Es hat sich in der Praxis immer wieder gezeigt, daß die Zeit, die für die Planung aufgewendet wird und die Zeit die für die Fehlersuche gebraucht wird im umgekehrten Verhältnis zueinander stehen. Gewiß, es gibt Programmiergenies, die schreiben Programme von 1/2 KByte frei herunter. Doch das sind sicher Ausnahmen.

In Bild 4.1.2 ist der Programmablaufplan für das Programm »STOPPUHR« dargestellt.

Im Flußdiagramm erhalten alle Variablen und Flags symbolische Namen. Aus diesen sollte die Bedeutung die diese Speicheradressen im Programm haben hervorgehen. Ein Punkt den man sich für seine eigenen Routinen merken sollte.
Anhand des fertigen Flußdiagramms stellen wir eine Variablenliste auf und legen die Pointerbasiswerte fest. Soll das Programm für eine andere Speicherkonfiguration modifiziert werden, so ist der Initialisierungsblock entsprechend zu ändern.
Die angegebenen absoluten Adressen wurden gewählt, weil sie im verwendeten Entwicklungssystem zur Verfügung standen.

Pointer 1 = $05E0_{16}$, Basisadresse des Displays und der Display-Kode-Tabelle sowie der Flags (Merker) und Zähler.
Pointer 2 = $042B_{16}$, Basisadresse für Sprünge und Speicherbefehle.
Die in Klammern aufgeführten Adressen sind alle in sedezimaler Darstellung geschrieben. Auf die Indizes wurde deshalb verzichtet.

DISTB (5E0): Tabelle die den Anzeigekode für die Ziffern 0 bis 9 und den Dezimalpunkt enthält.
DHSEK (5F0): Displayadresse hundertstel Sekunden
DZSEK (5F1): Displayadresse zehntel Sekunden
DSEKE (5F2): Displayadresse Sekunden-Einer
DSEKZ (5F3): Displayadresse Sekunden-Zehner
DMINE (5F4): Displayadresse Minuten-Einer
DMINZ (5F5): Displayadresse Minuten-Zehner
MIN (5ED): Minuten in BCD-Darstellung
SEK (5EC): Sekunden in BCD-Darstellung
HZSEK (5EB): hundertstel und zehntel Sekunden in BCD-Darstellung
ENDFL (5EE): Ende-Flag, signalisiert ob die Uhr gestoppt wurde.
COUNT (5EF): Zähler, gibt an welche der Anzeigestellen (Sekunden, Minuten) bedient werden sollen, bzw. ob ein Überlauf (Zeit größer 99 Minuten und 59,99 Sekunden) stattgefunden hat.
ZEITK (455): Zeitkonstante, dient zum Ausgleich verschiedener Programmlaufzeiten (je nach Verzweigung).

Selbstverständlich kann die gestellte Aufgabe noch besser gelöst werden. Die Programmbeispiele sind jedoch als Lern- bzw. Lehrbeispiele gedacht und deshalb übersichtlich strukturiert. Aus diesem Grund werden auch keine besonderen Programmiertricks angewendet.

Ermittlung der Zeitkonstanten und des Delay-Displacements im Programm »STOPPUHR«.

Es ist gefordert eine Basiszeiteinheit T = 10 ms = 0,01 s. Um eine Justierungsmöglichkeit über den Akkumulatorinhalt zu bekommen, soll mit dem Delay-Befehl eine Verzögerung von 9 ms = 0,009 s erzielt werden. Dazu ist jetzt das entsprechende Displacement zu berechnen.

$XX_{10} = (T_V(MZ) - 13_{10})/514_{10}$

1 MZ entspricht bei einer Taktfrequenz von 2 MHz 2 Mikrosekunden.
9 ms = 9000 µs = 4500 MZ

$XX_{10} = (4500 - 13)/514 = 8{,}73$

Als Displacement wird der Wert 9 gewählt.

Rechnet wieder in Mikrozyklen zurück, so erhält man 4626 MZ. Das entspricht einer Zeit von 9,252 ms.

Die zum geforderten Zeittakt fehlenden 0,748 ms werden über den Inhalt des Akkumulators erzeugt.

Der Akkuinhalt ist
$(AC)_{10} = ((T_V(MZ) - 13_{10})/2) - 257 \cdot XX_{10}$

Mit T_V = 10000 Mikrosekunden = 5000 MZ erhält man
$(AC)_{10} = (5000 - 13)/2 - 257 \cdot 9 = 180{,}5$ gerundet 180_{10}

Der Fehler durch die Rundung ist vernachlässigbar klein.
Der Akkumulator muß also mit $180_{10} = B4_{16}$ geladen werden bevor der Befehl DLY 09 ausgeführt wird.

Will man die Genauigkeit des quarzstabilisierten Systemtakts ausnutzen, muß man in der Regel schon etwas »probieren«.

Damit steht der Wert der Variablen ZEITK für den ersten Aufruf des Delay-Befehls fest. Im Programm sind jedoch noch drei Justierstellen vorgesehen. Abhängig von der Anzahl der abzuarbeitenden Instruktionen in einem Programmzweig wird die Variable ZEITK entsprechend verändert. Die Werte sind dem Listing des Programms selbst zu entnehmen.

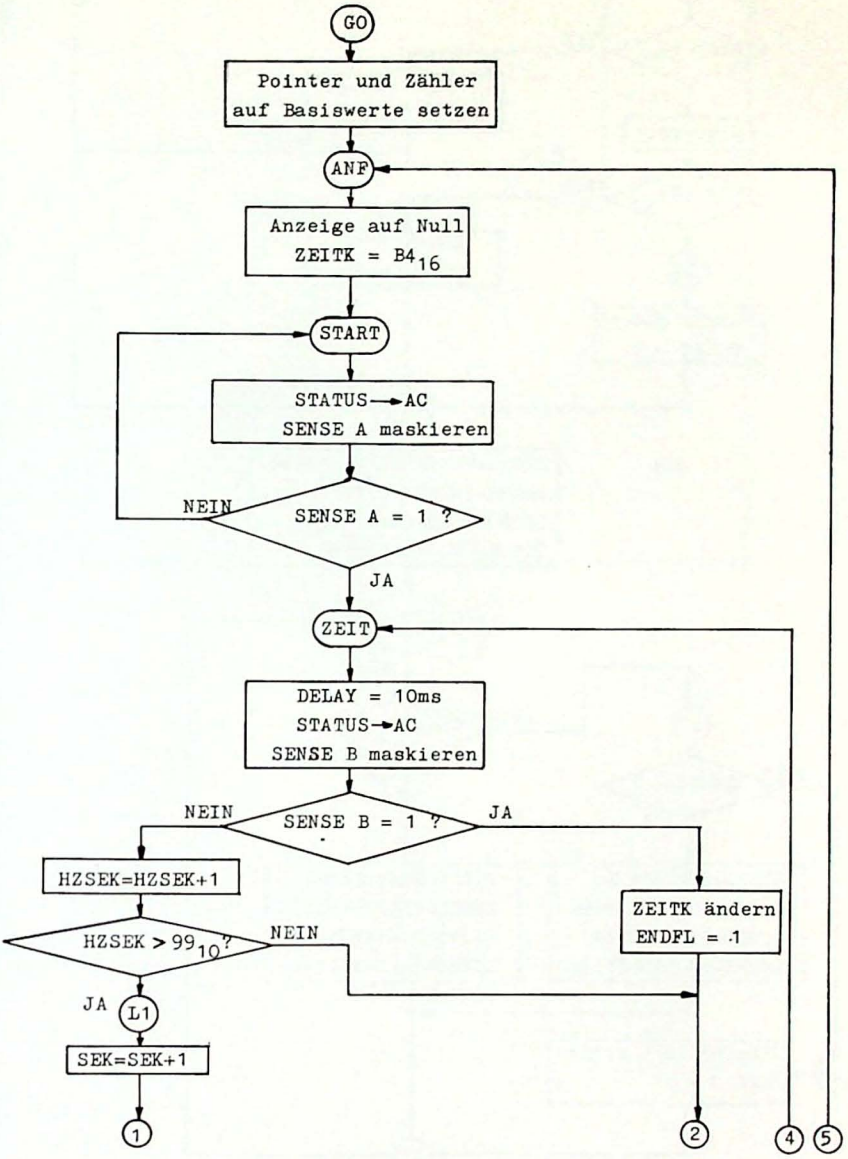


Bild 4.1.2: Flußdiagramm zum Programm »STOPPUHR«

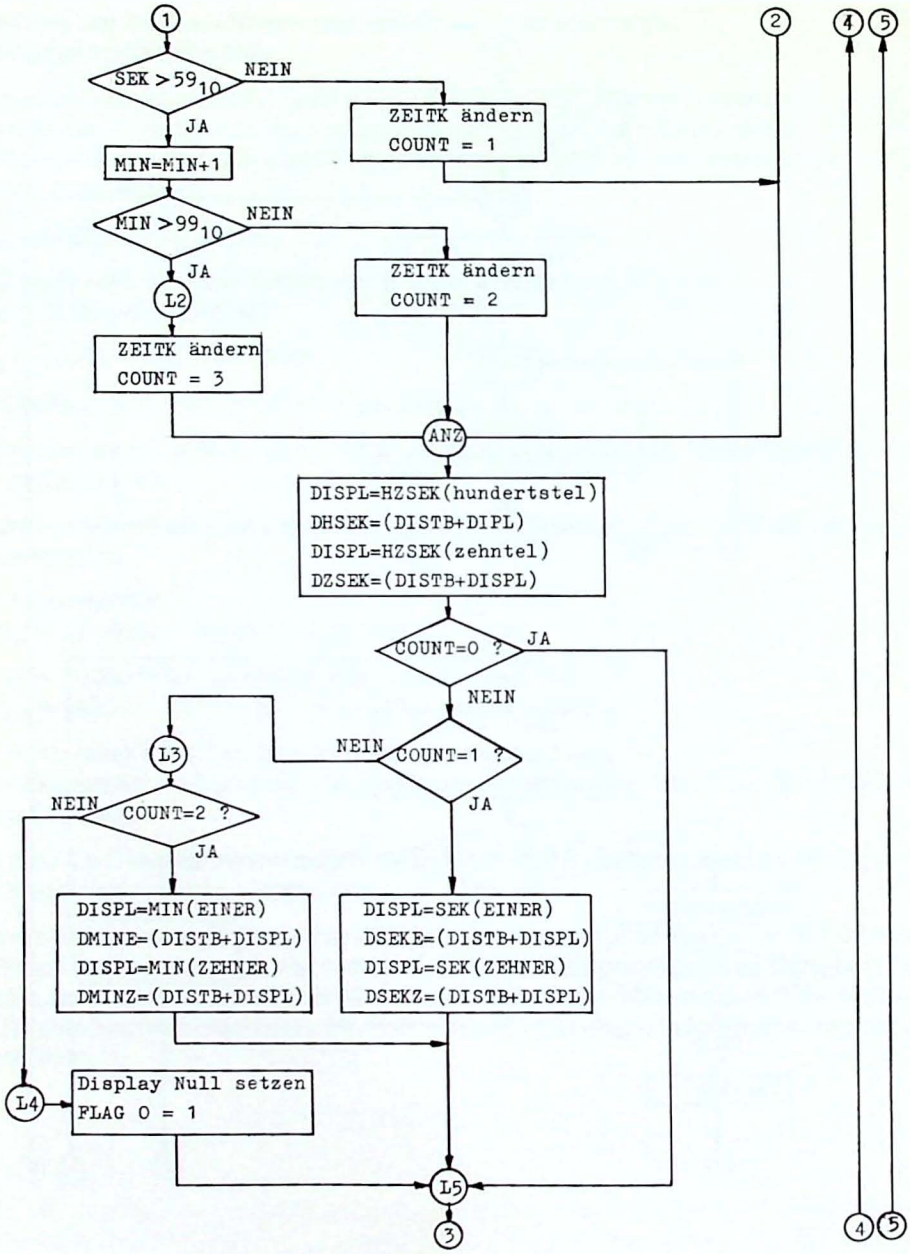


Bild 4.1.2: Fortsetzung

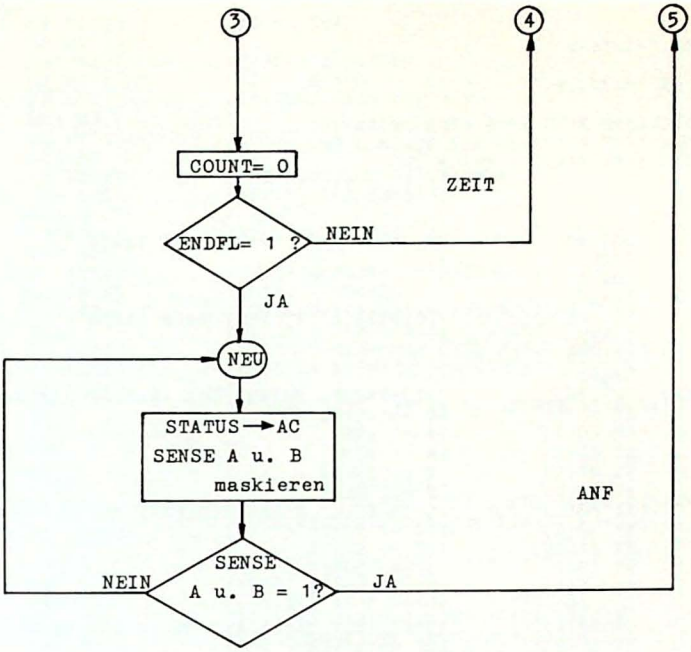


Bild 4.1.2: Fortsetzung

```
;
;
        .TITL STOPPUHR
;
;20.08.80 - REINHARD, M.
;
; STOPPUHR-PROGRAMM FUER SCMP MIKROPROZESSOR
; GUELTIG FUER EINE TAKTFREQUENZ VON 2,0 MHZ
; START/STOPP UEBER SENSE A UND B
;
.LOC 0420
;
0420 C405 GO:   LDI 05          ;POINTER 1 MIT WERT 05E0 LADEN
0422 35         XPAH1           ;
0423 C4E0       LDI E0          ;
0425 31         XPAL1           ;
0426 C404       LDI 04          ;POINTER 2 MIT WERT 042B LADEN
0428 36         XPAH2           ;
0429 C42B       LDI 2B          ;
042B 32         XPAL2           ;
042C C400 ANF:  LDI 00          ;VERWENDETE PUFFER UND ANZEIGE AUF 0 SETZEN
042E C90E       ST 0B(1)        ;0 => HZSEK
0430 C90C       ST 0C(1)        ;0 => SEK
0432 C90D       ST 0D(1)        ;0 => MIN
0434 C90E       ST 0E(1)        ;0 => ENDFL
0436 C90F       ST 0F(1)        ;0 => COUNT
0438 C100       LD 00(1)        ;CODE FUER "0"-ANZEIGE(3F) AUS DISTB LADEN
043A C910       ST 10(1)        ;3F => DHSEK
043C C911       ST 11(1)        ;3F => DZSEK
043E C913       ST 13(1)        ;3F => DSEKZ
0440 C915       ST 15(1)        ;3F => DMINZ
0442 C480       LDI 80          ;
0444 D900       OR 00(1)        ;CODE FUER DEZ.PUNKT UND 0 ADDIEREN
0446 C912       ST 12(1)        ;BF => DSEKE
0448 C914       ST 14(1)        ;BF => DMINE
044A C4F4       LDI F4          ;WERT FUER ZEITFEINEINSTELLUNG NACH
044C C808       ST ZEITK        ;ZEITK SPEICHERN
044E 04   START:DINT            ;DAMIT WIRKT SENSE A ALS EINFACHER EINGANG
044F 06         CSA             ;STATUS => AC
0450 D410       ANI 10          ;SENSE A AUSMASKIEREN
0452 98FA       JZ START        ;SPRUNG NACH START, SOLANGE SENSE A NICHT 1
0454 C400 ZEIT: LDI ZEITK       ;
0455 00   ZEITK:00              ;ZEITKONSTANTE
0456 8F09       DLY 09          ;10 MS VERZOEGERN
0458 06         CSA             ;STATUS => AC
0459 1420       ANI 20          ;SENSE B AUSMASKIEREN
045B 9804       JZ L0           ;SPRUNG NACH L0, WENN SENSE B NICHT BETAETIGT
045D C90E       ST 0E(1)        ;ENDFL = "1" SETZEN
045F 903C       JMP ANZ         ;SPRUNG ZUM ANZEIGETEIL
0461 02   L0:   CCL             ;CY/L DEFINIERT AUF NULL SETZEN
0462 C10B       LD 0B(1)        ;HZSEK => AC
0464 EC01       DAI 01          ;
0466 C90B       ST 0B(1)        ;HZSEK=HZSEK+1
0468 9C33       JNZ ANZ         ;ANZEIGE SOLANGE HZSEK <= 99.
046A C10C       LD 0C(1)        ;UEBERLAUF IN DIE SEKUNDEN IST ERFOLGT
046C EC00       DAI 00          ;SEK=SEK+1  (CY/L=1)
046E 01         XAE             ;SICHERN
046F C440       LDI 40          ;KONSTANTE FUER 40. LADEN
0471 68         DAE             ;SEK + 40.
0472 980D       JZ L1           ;SPRUNG, WENN SEK > 59.
0474 40         LDE             ;NEUEN SEKUNDENWERT NACH SEK SPEICHERN
0475 C90C       ST 0C(1)        ;
0477 C401       LDI 01          ;
0479 C90F       ST 0F(1)        ;COUNT = 1
047B C482       LDI 82          ;
047D C8F7       ST ZEITK        ;ZEITKONSTANTE JUSTIEREN
047F 901C       JMP ANZ         ;SPRUNG ZUR ANZEIGE
```

```
04E1 C90C L1:   ST 0C(1)        ;00 => SEK
04E3 C10D       LD 0D(1)        ;
04E5 EC00       DAI 00          ;CY/L=1 ADDIEREN
04E7 C90D       ST 0D(1)        ;MIN = MIN+1
04E9 9E0A       JZ L2           ;SPRUNG, WENN MIN > 99.
04EB C402       LDI 02          ;
04ED C90F       ST 0F(1)        ;COUNT = 2
04EF C4E0       LDI 80          ;
04F1 C8C3       ST ZEITK        ;ZEITKONSTANTE JUSTIEREN
04F3 9008       JMP ANZ         ;SPRUNG ZUR ANZEIGE
04F5 C403 L2:   LDI 03          ;
04F7 C90F       ST 0F(1)        ;COUNT = 3
04F9 C4B0       LDI B0          ;
04FB C8B9       ST ZEITK        ;ZEITKONSTANTE JUSTIEREN
04FD C10B ANZ:  LD 0B(1)        ;ANZEIGE-LOGIK
04FF D40F       ANI 0F          ;HUNDERTSTEL EINER AUSMASKIEREN
0501 01         XAE             ;EINER IM EXT. REG. SICHERN
0502 C180       LD 80(1)        ;DISPLAY-CODE MIT <EXT.REG.> ALS DISPL. LADEN
0504 C910       ST 10(1)        ;CODE NACH DHSEK SCHREIBEN
0506 C10B       LD 0B(1)        ;
0508 1C         SR              ;HUNDERTSTEL ZEHNER IN RECHTE BYTE-HAELFTE
0509 1C         SR              ;
050A 1C         SR              ;
050B 1C         SR              ;
050C 01         XAE             ;ZEHNER => EXT.REG.
050D C180       LD 80(1)        ;DISPL.-CODE MIT <EXT.> ALS DISPL. LADEN
050F C911       ST 11(1)        ;CODE NACH DZSEK SCHREIBEN
0511 C10F       LD 0F(1)        ;
0513 9E47       JZ L5           ;SPRUNG, WENN COUNT = 0
0515 B90F       DLD 0F(1)       ;COUNT = COUNT-1
0517 9C18       JNZ L3          ;SPRUNG, WENN COUNT > 1 WAR
0519 C10C       LD 0C(1)        ;ANZEIGE DER SEKUNDEN ERFORIERLICH
051B D40F       ANI 0F          ;SEKUNDEN-EINER AUSMASKIEREN
051D 01         XAE             ;UND ALS DISPLACEMENT SICHERN
051E C180       LD 80(1)        ;DISPLAY-CODE LADEN
0520 DC80       ORI 80          ;DEZIMALPUNKT ADDIEREN
0522 C912       ST 12(1)        ;CODE NACH DSEKE SCHREIBEN
0524 C10C       LD 0C(1)        ;SEKUNDEN-ZEHNER IN RECHTE BYTE-HAELFTE BRINGEN
0526 1C         SR              ;
0527 1C         SR              ;
0528 1C         SR              ;
0529 1C         SR              ;
052A 01         XAE             ;ALS DISPL. SICHERN
052B C180       LD 80(1)        ;CODE LADEN
052D C913       ST 13(1)        ;NACH DSEKZ SCHREIBEN
052F 902B       JMP L5          ;
0531 B90F L3:   DLD 0F(1)       ;COUNT = COUNT-1
0533 9C18       JNZ L4          ;SPRUNG, WENN COUNT > 2 WAR
0535 C10D       LD 0D(1)        ;
0537 D40F       ANI 0F          ;MINUTEN-EINER AUSMASKIEREN
0539 01         XAE             ;ALS DISPL. SICHERN
053A C180       LD 80(1)        ;DISPLAY-CODE LADEN
053C DC80       ORI 80          ;DEZIMALPUNKT ADDIEREN
053E C914       ST 14(1)        ;CODE NACH DMINE SCHREIBEN
0540 C10D       LD 0D(1)        ;
0542 1C         SR              ;MINUTEN-ZEHNER IN RECHTE BYTE-HAELFTE BRINGEN
0543 1C         SR              ;
0544 1C         SR              ;
0545 1C         SR              ;
0546 01         XAE             ;ALS DISPL. SICHERN
0547 C180       LD 80(1)        ;DISPL.CODE LADEN
0549 C915       ST 15(1)        ;NACH DMINZ SCHREIBEN
054B 900F       JMP L5          ;
054D C100 L4:   LD 00(1)        ;UEBERLAUF, ZEIT > 99MIN,59,99SEK
054F C913       ST 13(1)        ;RESTLICHE ANZEIGESTELLEN AUF "0  SETZEN
0551 C915       ST 15(1)        ;
0553 DC80       ORI 80          ;DEZIMALPUNKT ADDIEREN
```

```
04F5 C912        ST 12(1)          ;
04F7 C914        ST 14(1)          ;
04F9 C401        LDI 01            ;FLAG 0 = 1 SETZEN, UEBERLAUFANZEIGE
04FB 07          CAS               ;
04FC C400 L5:    LDI 00            ;COUNT = 0 SETZEN
04FE C90F        ST 0F(1)          ;
0500 C10E        LD 0E(1)          ;
0502 9A28        JZ 28(2)          ;SPRUNG NACH ZEIT, WENN ENDFL=0
0504 06    NEU:  CSA               ;TEST SENSE A UND B GESETZT
0505 E430        XRI 30            ;
0507 9CFB        JNZ NEU           ; NICHT GESETZT
0509 07          CAS               ;FLAG 0 = 0 SETZEN
050A 9200        JMP 00(2)         ;SPRUNG NACH ANF
;
;
.END
```

4.2 Geschwindigkeitsmessung mit dem Programm »GESCHWMESS«

Ähnlich wie im ersten Beispiel geht es auch hierbei um die Erfassung von Zeiten. Es wird die Zeit gemessen die ein Fahrzeug oder eine Person für die Zurücklegung einer bestimmten Wegstrecke benötigt. Bei bekannter Strecke läßt sich über die Beziehung v = s/t die Geschwindigkeit berechnen.

Dabei gilt:
v = Geschwindigkeit in Meter/Sekunde (m/s)
t = Zeit in Sekunden (s)
s = Weg in Meter (m)

Da uns eine Geschwindigkeitsangabe in m/s im Allgemeinen recht wenig sagt, wird diese in km/h berechnet.

Die Meßstrecke soll eine Länge von 1,0 m haben.

Die Messung beginnt mit einem »0/1«-Wechsel am SENSE B-Eingang. Sie endet, wenn am Eingang SENSE A eine »1« erscheint.

In diesem Programm soll die Interruptmöglichkeit des SCMP ausgenutzt werden. Auf die Ansteuerung eines Displays wird verzichtet. In den Speicherstellen mit den symbolischen Adressen ERG(H) und ERG(L) steht das Ergebnis als vorzeichenlose 16-Bit Binärzahl zur weiteren Verarbeitung zur Verfügung.

Vor Erstellung des Flußdiagramms müssen die mathematischen Zusammenhänge geklärt werden. Es muß ein Algorithmus gefunden werden, der uns den Weg von der gemessenen Zeit zur Geschwindigkeit in km/h zeigt.

Es gilt: 1 m/s = 3,6 km/h

Legen wir die Meßzeiteinheit auf 1,5 ms fest, so erhalten wir eine Konstante K = 3,6/0,0015 = 2400. Dividieren wir diese Konstante durch die Anzahl der Zeiteinheiten, so erhalten wir die Geschwindigkeit in km/h. Bei der Berechnung mit dem Mikroprozessor haben wir es dadurch nur mit ganzen Zahlen zu tun.

Die kleinste noch meßbare Geschwindigkeit ist 9 km/h, die größte ist 2400 km/h. Im Bereich 80 km/h muß mit einem Fehler von ca. 3,5% gerechnet werden.

Der Aufbau dieses Programms wurde gewählt um die Interruptverarbeitung mit dem SCMP zu zeigen. Die Struktur stellt ebenso wie der Rücksprung aus dem Unterprogramm sicher nicht das Optimum dar. Das ist aber auch gar nicht der Sinn dieses Programms. Worauf es ankommt ist zu zeigen wie man mit dem SCMP Unterbrechungen bearbeiten kann und was dabei zu beachten ist.

Den schwierigsten Teil des Programms bildet die Division. Dieser Programmabschnitt kann bei Bedarf zu einem »stand alone«-Divisionsprogramm modifiziert werden. Auch ist die Erweiterung auf zum Beispiel 32 Bit und mehr möglich. Es müssen dazu alle Variablen auf vier Byte erweitert werden. Ausgenommen davon sind selbstverständlich die Zähler. Außerdem sind verschiedene Schleifendurchläufe zu ändern.

Bei der Division wird zuerst der Divisor solange nach links geschoben (Multiplikation mit 2) bis er unter dem höchstwertigen Bit des Dividenden steht. Dabei wird der Zähler »N« hochgezählt. Jetzt wird der Divisor solange vom Dividenden abgezogen und dabei der Zähler »N« dekrementiert bis N = 0 erreicht ist. In den Bytes ERG(H) und ERG(L) steht danach das Ergebnis der Division und damit die Geschwindigkeit in km/h.

Doch noch einmal zurück zur Division. Tritt bei ihr ein Rest auf, so steht dieser in der gleichlautenden Variablen REST(H) und REST(L). Liefert bei der Rechnung die Subtraktion des Teilers vom Dividenden ein negatives Ergebnis, so wird eine »0« in den Quotienten geschrieben. Ist es positiv, so erhält der Quotient eine »1«. Der Quotient selbst wird beginnend beim höchstwertigen Bit gebildet.

Zur Kontrolle bzw. zur Steuerung von Hardware werden über die Flags 0, 1 und 2 die »Betriebszustände« des Programms »GESCHWMESS« angezeigt.

Die Flags und ihre Bedeutung:

Flag 0 = 1 : Bereitschaft
Flag 1 = 1 : Rechnung wird ausgeführt
Flag 2 = 1 : Messung
Flag 0, 1, 2 = 1: Überlauf (Geschwindigkeit kleiner 9 km/h)
Flag 0, 1, 2 = 0: Ergebnis steht bereit

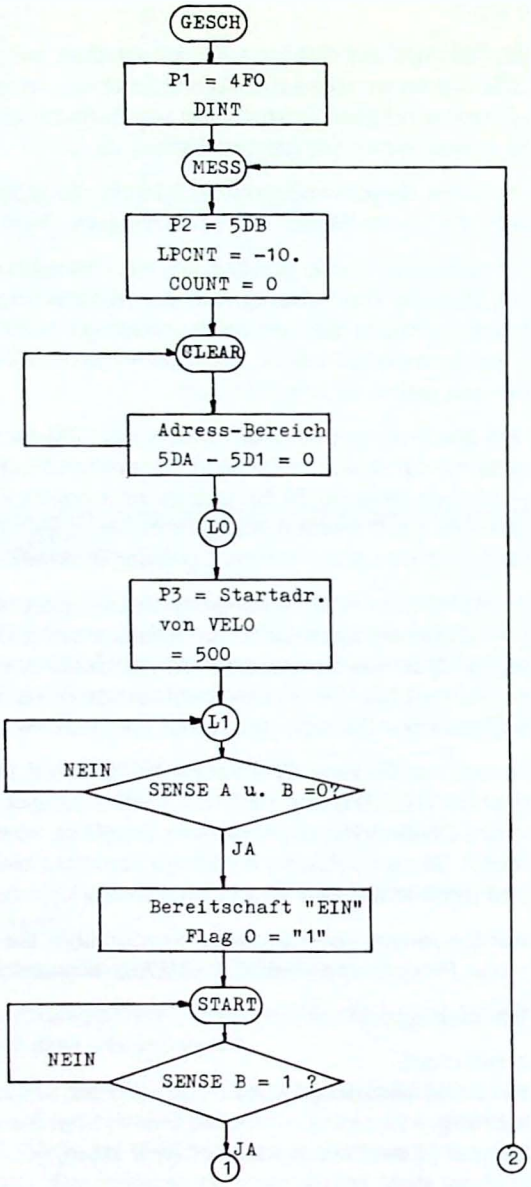


Bild 4.2.1: Flußdiagramm zum Programm

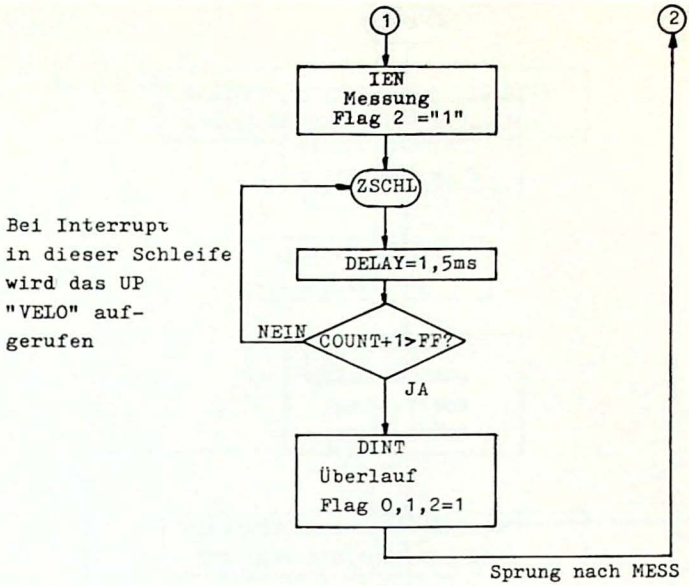


Bild 4.2.1: Fortsetzung

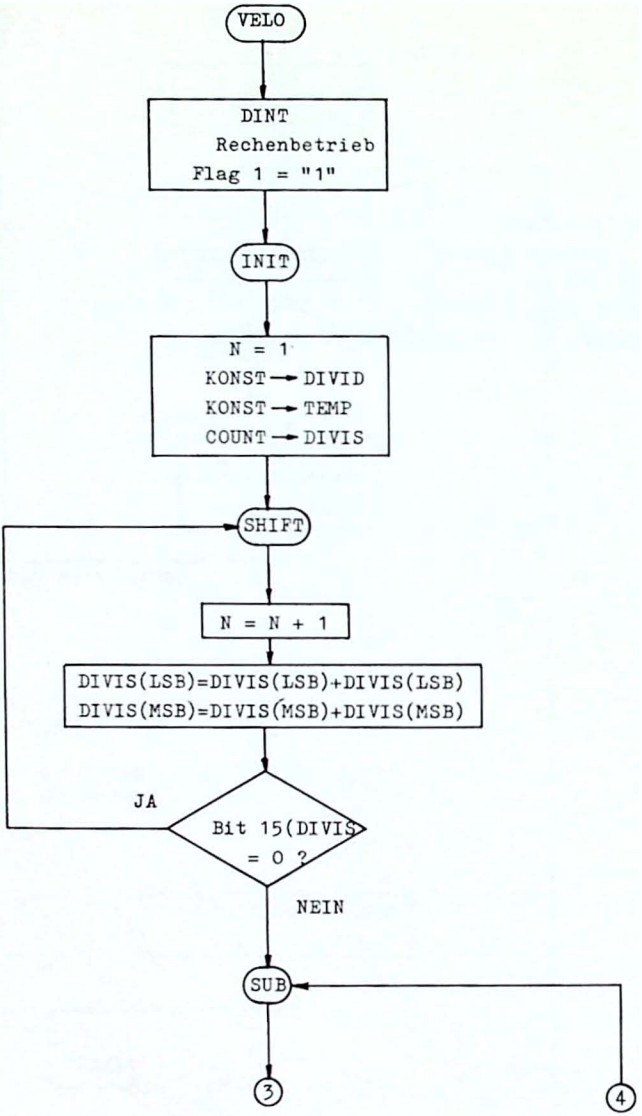


Bild 4.2.2: Flußdiagramm zum Interrupt-UP »VELO«

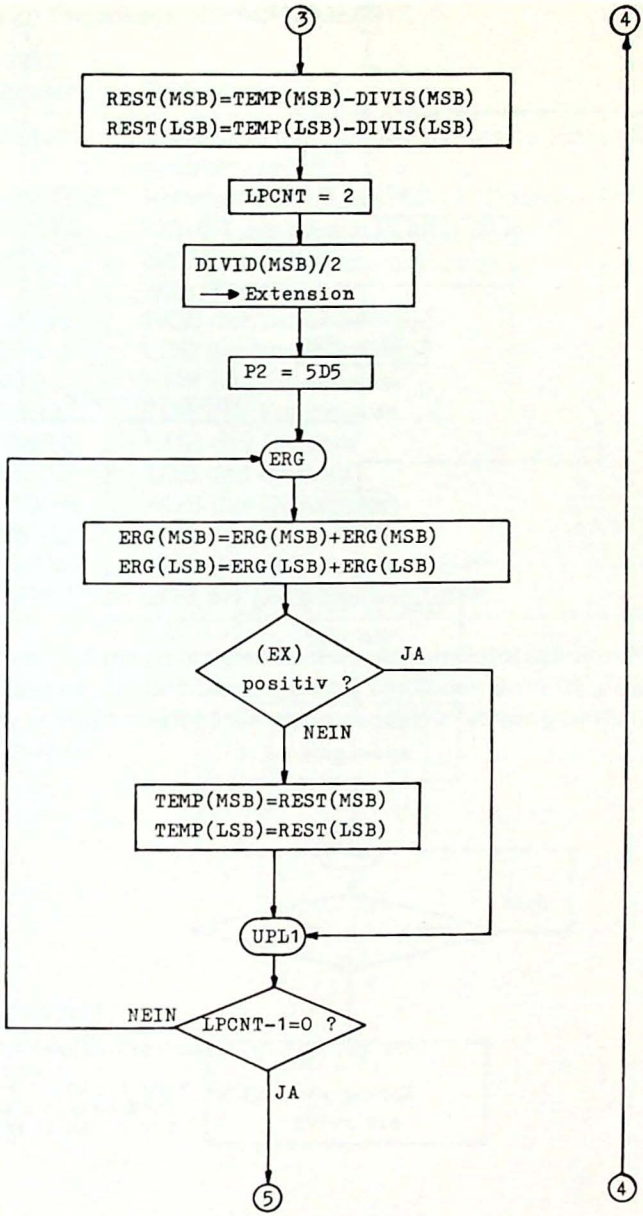


Bild 4.2.2: Fortsetzung

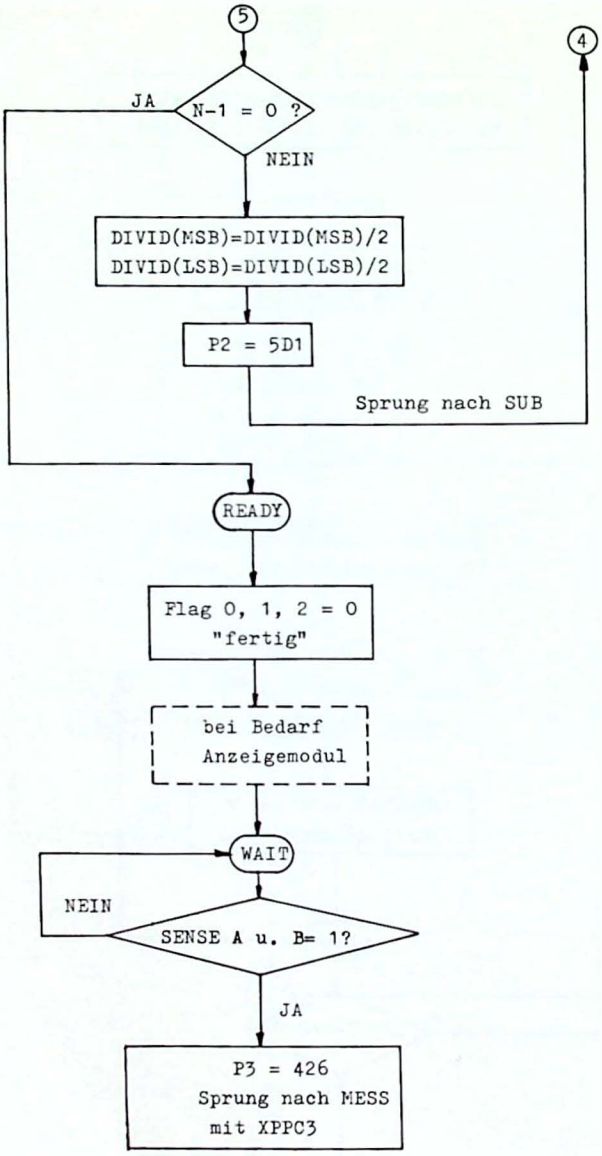


Bild 4.2.2: Fortsetzung

Variablenliste zum Programm »GESCHWMESS«

absolute Adresse	symbol. Adresse		Bedeutung
$4F0_{16}$	COUNT	;	Zähler mit dem die Zeitbasistakte eines Meßzyklus gezählt werden.
$4F1_{16}$	KONST(H)	;	Konstante $2400_{10} = 960_{16}$ in Doppelwortdarstellung
$4F2_{16}$	KONST(L)	;	KONST wird durch COUNT dividiert
$4F3_{16}$	LPCNT	;	Schleifenzähler
$4F4_{16}$	N	;	Hilfszähler
$5D1_{16}$	REST(H)	;	HOB der Variablen REST
$5D2_{16}$	REST(L)	;	LOB der Variablen REST
$5D3_{16}$	ERG(H)	;	HOB des Ergebnisses
$5D4_{16}$	ERG(L)	;	LOB des Ergebnisses
$5D5_{16}$	DIVIS(H)	;	HOB des Divisors
$5D6_{16}$	DIVIS(L)	;	LOB des Divisors
$5D7_{16}$	DIVID(H)	;	HOB des Dividenden
$5D8_{16}$	DIVID(L)	;	LOB des Dividenden
$5D9_{16}$	TEMP(H)	;	HOB der Hilfsvariablen TEMP
$5DA_{16}$	TEMP(L)	;	LOB der Hilfsvariablen TEMP

Um den Zeittakt von 1,5 ms zu erzielen wird der Akkumulator mit dem Wert 70_{16} geladen und das Displacement des Delay-Befehls erhält den Wert 01_{16}. Auf die Berechnung kann an dieser Stelle verzichtet werden, da wir den Vorgang ausführlich in Kapitel 4.1 beschrieben haben.

```
;
;
        .TITL GESCHWMESS
;
; PROGRAMM ZUR GESCHWINDIGKEITSMESSUNG MIT DEM SCMP
; SYSTEMTAKT = 2MHZ
; KLEINSTE GESCHW. = CA. 9 KM/H
; GROESSTE GESCHW. = 2400 KM/H
; FEHLER BEI 80 KM/H CA. 3,5 %
;
;
.LOC 0420
;
0420 C404 GESCH:LDI 04          ;START HAUPTPROGRAMM
0422 35         XPAH1           ;
0423 C4F0       LDI F0          ;
0425 31         XPAL1           ;POINTER 1 = 04F0
0426 04         DINT            ;INTERRUPTS SPERREN
```

```
0427 C405 MESS: LDI 05          ;
0429 36         XPAH2           ;
042A C4DB       LDI DB          ;
042C 32         XPAL2           ;POINTER 2 = 05DB
042D C4F6       LDI F6          ;
042F C903       ST 03(1)        ;LPCNT = F6 = -10.
0431 C400       LDI 00          ;
0433 C900       ST 00(1)        ;COUNT = 00
0435 CEFF CLEAR:ST @FF(2)       ;0 => 5DA-5D1
0437 A903       ILD 03(1)       ;LPCNT = LPCNT + 1
0439 9804       JZ L0           ;SPRUNG, WENN ALLE PUFFER GELOESCHT SIND
043B C400       LDI 00          ;
043D 90F6       JMP CLEAR       ;NEUE SCHLEIFE
043F 33   L0:   XPAL3           ;
0440 C405       LDI 05          ;
0442 37         XPAH3           ;P3 = 500, START-ADRESSE VON "VELO"
0443 06   L1:   CSA             ;
0444 D430       ANI 30          ;SENSE A UND B AUSMASKIEREN
0446 9CFB       JNZ L1          ;SPRUNG, WENN A UND B NICHT 0
0448 C401       LDI 01          ;
044A 07         CAS             ;FLAG 0 = 1, "BEREITSCHAFT"
044B 06   START:CSA             ;SENSE B ABFRAGEN
044C D420       ANI 20          ;
044E 98FB       JZ START        ;WARTESCHLEIFE, SOLANGE SENSE B NICHT "1"
0450 C40C       LDI 0C          ;
0452 07         CAS             ;INTERRUPT ERLAUBEN UND FLAG 2=1, "MESSUNG"
0453 C470 ZSCHL:LDI 70          ;
0455 8F01       DLY 01          ;DELAY = 1,5 MS
0457 A900       ILD 00(1)       ;COUNT = COUNT-1
0459 9CF8       JNZ ZSCHL       ;SCHLEIFE BIS INTERRUPT ODER UEBERLAUF
045B C407       LDI 07          ;INTERRUPT SPERREN, FLAG 0,1,2 = 1, "UEBERLAUF
045D 07         CAS             ;
045E 90C0       JMP MESS        ;RUECKSPRUNG NACH "MESS"
```

```
;
; UNTERPROGRAMM "VELO"
; BERECHNET DIE GESCHWINDIGKEIT
;
.LOC 0500
;
0500 08   VELO: NOP             ;
0501 04         DINT            ;WEITERE INTERRUPTS SPERREN
0502 C402       LDI 02          ;
0504 07         CAS             ;FLAG 1 = 1, "RECHNUNG"
0505 C401 INIT: LDI 01          ;
0507 C904       ST 04(1)        ;N = 1
0509 C101       LD 01(1)        ;
050B CA06       ST 06(2)        ;DIVID(H) = KONST(H)
050D CA08       ST 08(2)        ;TEMP(H) = KONST(H)
050F C102       LD 02(1)        ;
0511 CA07       ST 07(2)        ;DIVID(L) = KONST(L)
0513 CA09       ST 09(2)        ;TEMP(L) = KONST(L)
0515 C100       LD 00(1)        ;
0517 CA05       ST 05(2)        ;DIVIS(L) = COUNT
0519 A904 SHIFT:ILD 04(1)       ;N = N + 1
051B C205       LD 05(2)        ;
051D F205       ADD 05(2)       ;DIVIS(L)=DIVIS(L)*2
051F CA05       ST 05(2)        ;
0521 C204       LD 04(2)        ;
0523 F204       ADD 04(2)       ;
0525 CA04       ST 04(2)        ;DIVIS(H)=DIVIS(H)*2
0527 94F0       JP SHIFT        ;SPRUNG, SOLANGE DAS ERGEBNIS POSITIV IST
```

```
0529 C209 SUB:   LD 09(2)          ;
052B 03          SCL               ;
052C FA05        CAD 05(2)         ;
052E CA01        ST 01(2)          ;REST(L) = TEMP(L)-DIVIS(L)
0530 C208        LD 08(2)          ;
0532 FA04        CAD 04(2)         ;
0534 CA00        ST 00(2)          ;REST(H) = TEMP(H)-DIVIS(H)
0536 C402        LDI 02            ;
0538 C903        ST 03(1)          ;LPCNT = 2
053A C206        LD 06(2)          ;DIVID(H)/2 => EXT.REG.
053C 1D          SRL               ;
053D 01          XAE               ;
053E C604        LD @04(2)         ;P2 = 05D5
0540 C2FF ERG:   LD FF(2)          ;
0542 F2FF        ADD FF(2)         ;
0544 CEFF        ST @FF(2)         ;ERG(L,H) = ERG(L,H)+ERG(L,H)
0546 40          LDE               ;P2 HAT WERTE 5D4,5D3
0547 9404        JP UPI1           ;SPRUNG, SOLANGE CY/L = 0
0549 C2FE        LD FE(2)          ;
054B CA06        ST 06(2)          ;TEMP(L,H) = REST(L,H)
054D B903 UPL1:  DLD 03(1)         ;LPCNT = LPCNT-1
054F 9CEF        JNZ ERG           ;SCHLEIFE FUER LSP UND MSP
0551 B904        DLD 04(1)         ;N = N-1
0553 980E        JZ READY          ;WENN 0, STEHT ERGEBNIS IN ERG(L,H)
0555 C202        LD 02(2)          ;DIVIS DURCH 2 TEILEN
0557 1F          RRL               ;
0558 CA02        ST 02(2)          ;DIVIS(H) = DIVIS(H)/2
055A C203        LD 03(2)          ;
055C 1F          RRL               ;
055D CA03        ST 03(2)          ;DIVIS(L) = DIVIS(L)/2
055F C6FE        LD @FE(2)         ;P2 = 5D1
0561 90C6        JMP SUB           ;
0563 C400 READY:LDI 00             ;
0565 07          CAS               ;FLAG 0,1,2 = 0, ERGEBNIS VORHANDEN
0566 06    WAIT: CSA               ;HIER KOENNTE ANZEIGEMODUL FOLGEN
0567 E430        XRI 30            ;SENSE A UND B GESETZT ?
0569 9CFB        JNZ WAIT          ;SPRUNG, WENN NEIN
056B C404        LDI 04            ;JA, RUECKSPRUNG INS HAUPTPROGRAMM
056D 37          XPAH3             ;
056E C426        LDI 26            ;
0570 33          XPAL3             ;P3 = 0426
0571 3F          XPPC3             ;RUECKSPRUNG
;
;
.END
```

4.3 Das Programm »DIGITALUHR«

Im dritten Beispielprogramm wollen wir uns noch einmal mit der Zeit beschäftigen. Ähnlich wie beim Stoppuhrenprogramm werden auch hier Zeiteinheiten summiert. Zur Anzeige werden Stunden und Minuten gebracht. Die Sekunden werden in Form einer blinkenden Leuchtdiode, die über Flag 0 angesteuert wird, dargestellt. Dieses Mal wird die Anzeige von BCD zu Siebensegment-Dekoder-/Treiberbausteinen angesteuert. Die Bausteine werden behandelt als seien sie Speicherstellen. Man bezeichnet diese Art der Ausgabe auch als »memory mapped output«. Selbstverständlich kann die Methode auch für eine Eingabe benutzt werden.

Für das Programm hat die gewählte Ausgabeform den Vorteil, daß jede Anzeigestelle direkt mit der BCD dargestellten Ziffer angesteuert werden kann.

Es bleibt dem Leser überlassen, zusätzlich auch noch die Sekunden auf diese Weise anzuzeigen.

Gestellt wird die »Uhr«, in dem man die gewünschte Startzeit in BCD-Form in die symbolischen Adressen BCDSE, BCDMI und BCDST schreibt und danach das Programm zu diesem Zeitpunkt startet.

Variablenliste:

absolute Adresse	symbolische Adresse	Bedeutung
500_{16}	BCDSE	Sekundenzähler in BCD-Darstellung
501_{16}	BCDMI	Minutenzähler in BCD-Darstellung
502_{16}	BCDST	Stundenzähler in BCD-Darstellung
503_{16}	DMINE	Anzeigestelle Minuten-Einer
504_{16}	DMINZ	Anzeigestelle Minuten-Zehner
505_{16}	DSTDE	Anzeigestelle Stunden-Einer
506_{16}	DSTDZ	Anzeigestelle Stunden-Zehner

Die unterschiedlichen Akkumulatorinhalte vor den Delay-Befehlen sind damit zu erklären, daß in einem Fall bei der Umwandlung von Dezimal in Sedezimal aufgerundet und im anderen Fall abgerundet wird.

Bei dem relativ einfachen Programm erübrigt sich eine weitergehende Erläuterung.

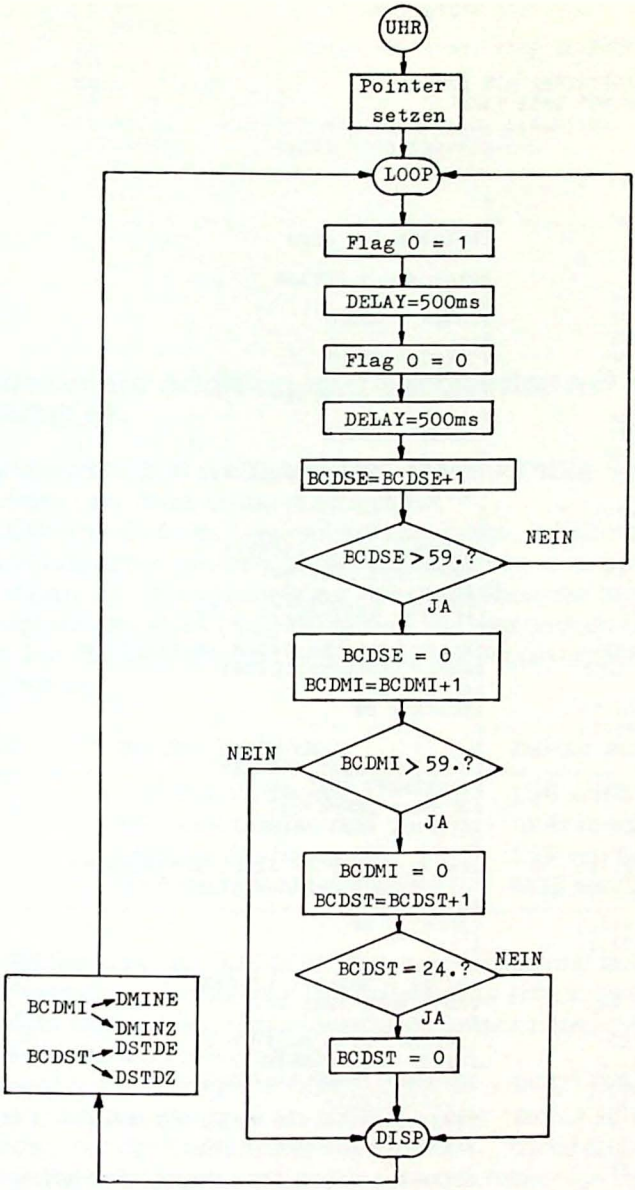


Bild 4.3.1: Flußdiagramm zum Programm »DIGITALUHR«

```
;
;
        .TITL DIGITALUHR
;
; SIMULATION EINER DIGITALUHR MIT DEM
; MIKROPROZESSOR SCMP MIT 2MHZ TAKT
;
.LOC 0420
;
0420 C405 UHR:  LDI 05  ;
0422 35         XPAH1           ;
0423 C400       LDI 00          ;
0425 31         XPAL1           ;POINTER 1 = 0500
0426 C401 LOOP: LDI 01  ;
0428 07         CAS             ;FLAG 0 = 1 SETZEN
0429 C42A       LDI 2A          ;
042B 8FF3       DLY F3          ;DELAY = 250MS
042D C42B       LDI 2B          ;
042F 8FF3       DLY F3          ;DELAY = 250MS
0431 C400       LDI 00          ;
0433 07         CAS             ;FLAG 0 = 0 SETZEN
0434 C42A       LDI 2A          ;
0436 8FF3       DLY F3          ;DELAY = 250MS
0438 C42B       LDI 2B          ;
043A 8FF3       DLY F3          ;DELAY = 250MS, GES.ZEIT = 1,0 SEK
043C 02         CCL             ;CY/L = 0
043D C100       LD 00(1)        ;
043F EC01       DAI 01          ;
0441 01         XAE             ;BCDSE=BCDSE+1 SICHERN
0442 C440       LDI 40          ;KONSTANTE FUER 40.
0444 68         DAE             ;BCDSE+40.
0445 9805       JZ L1           ;SPRUNG, WENN BCDSE > 59.
0447 40         LDE             ;
0448 C900       ST 00(1)        ;NEUEN WERT VON BCDSE ABSPEICHERN
044A 90DA       JMP LOOP        ;NEUE SEKUNDENSCHLEIFE
044C 02   L1:   CCL             ;CY/L = 0
044D C900       ST 00(1)        ;BCDSE = 00
044F C101       LD 01(1)        ;
0451 EC01       DAI 01          ;
0453 01         XAE             ;BCDMI=BCDMI+1 SICHERN
0454 C440       LDI 40          ;KONSTANTE FUER 40.
0456 68         DAE             ;BCDMI = BCDMI+40.
0457 9805       JZ L2           ;SPRUNG, WENN BCDMI > 59.
0459 40         LDE             ;
045A C901       ST 01(1)        ;NEUEN WERT FUER BCDMI ABSPEICHERN
045C 9010       JMP DISP        ;SPRUNG ZUR ANZEIGELOGIK
045E 02   L2:   CCL             ;CY/L = 0
045F C901       ST 01(1)        ;BCIMI = 00
0461 C102       LD 02(1)        ;
0463 EC01       DAI 01          ;
0465 01         XAE             ;BCDST = BCDST+1 SICHERN
0466 C424       LDI 24          ;KONSTANTE FUER 24.
0468 60         XRE             ;
0469 9801       JZ L3           ;SPRUNG, WENN BCDST = 24.
046B 40         LDE             ;NEUES BCDST LADEN
046C C902 L3:   ST 02(1)        ;00 ODER BCDST ABSPEICHERN
046E C101 DISP: LD 01(1)        ;ANZEIGELOGIK
0470 D40F       ANI 0F          ;ZEHNERSTELLE VON BCDMI LOESCHEN
0472 C903       ST 03(1)        ;EINER IN DISPLAY DMINE
0474 C101       LD 01(1)        ;
0476 1C         SR              ;ZEHNER VON BCDMI IN RECHTE BYTE HAELFTE
0477 1C         SR              ;
0478 1C         SR              ;
0479 1C         SR              ;
047A C904       ST 04(1)        ;ZEHNER NACH DMINZ SCHREIBEN
047C C102       LD 02(1)        ;
047E D40F       ANI 0F          ;ZEHNER VON BCDST LOESCHEN
```

```
0480 C905       ST 05(1)        ;EINER NACH DSTDE SCHREIBEN
0482 C102       LD 02(1)        ;
0484 1C         SR              ;ZEHNER IN RECHTE BYTE HAELFTE
0485 1C         SR              ;
0486 1C         SR              ;
0487 1C         SR              ;
0488 C906       ST 06(1)        ;ZEHNER NACH DSTDZ SCHREIBEN
048A 9097       JMP LOOP        ;NEUE SEKUNDENSCHLEIFE
;
;
.END
```

4.4 Sedezimale Addition und Subtraktion mit dem Programm »XADSUB«

Beim Programm XADSUB (HE**X** **AD**DITION **SUB**TRAKTION) wird eine Doppelwort-(16 Bit)-Addition bzw. Subtraktion durchgeführt.
Wird beim Aufruf eine »0« im Akkumulator übergeben, so bedeutet dies Addition. Bei einer »1« soll subtrahiert werden. In diesem Programm wird, um auch diese Technik einmal zu zeigen, die Stack-Technik zur Parameterübergabe benutzt.
Pointer 2 zeigt auf den Stack (Stapelspeicher) und wird deshalb auch als Stackpointer bezeichnet. Für die Organisation des Stacks vor Programmaufruf gilt die nachfolgende Aufstellung.

Stackpointer	bei der Subtraktion	bei der Addition
+ 0	LSB des Minuenden	LSB von Summand 1
+ 1	MSB des Minuenden	MSB von Summand 1
+ 2	LSB des Subtrahenden	LSB von Summand 2
+ 3	MSB des Subtrahenden	MSB von Summand 2

Nach dem Rücksprung aus XADSUB zeigt der Stackpointer auf das niederwertige Byte der Differenz bzw. der Summe. Der Akkumulator enthält den Wert des CY/L-Bits und des OV-Bits. Damit ist es möglich eventuelle Bereichsüberschreitungen im Hauptprogramm entsprechend zu verarbeiten.
Aufgerufen wird das Hilfsprogramm XADSUB mit dem XPPC3-Befehl. In XADSUB wird angenommen, daß der Stackpointer auf 500_{16} steht. Nach Ausführung der Addition bzw. Subtraktion hat der Stackpointer den Wert 504_{16}. An der Adresse 504_{16} steht das niederwertige Byte der Ergebnisse, in 505_{16} das höchstwertige Byte. Dies entspricht nicht ganz der allgemein üblichen Stacktechnik, hat aber den Vorteil, daß die Operanden nicht zerstört werden.
Nachfolgend eine Aufstellung der verwendeten symbolischen Adressen des Flußdiagramms.

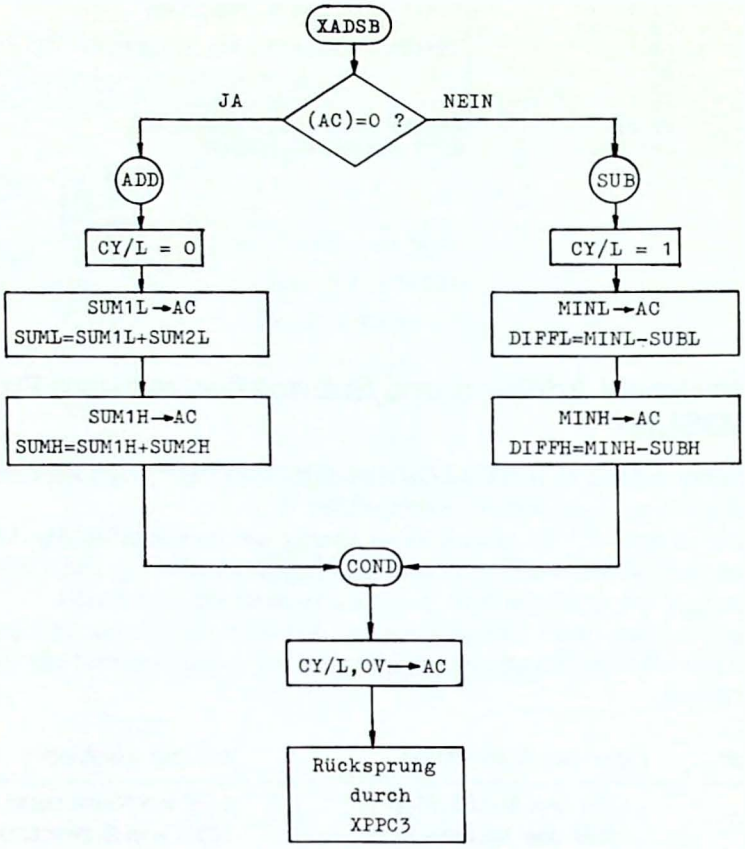


Bild 4.4.1: Flußdiagramm zum Programm »XADSUB«

Adresse	symbolische Adresse	Funktion
500_{16}	SUM1L oder MINL	LSB Summand 1 oder LSB Minuend
501_{16}	SUM1H oder MINH	MSB Summand 1 oder MSB Minuend
502_{16}	SUM2L oder SUBL	LSB Summand 2 oder LSB Subtrahend
503_{16}	SUM2H oder SUBH	MSB Summand 2 oder MSB Subtrahend
504_{16}	SUML oder DIFFL	LSB der Summe oder LSB der Differenz
505_{16}	SUMH oder DIFFH	MSB der Summe oder MSB der Differenz

```
;
;
        .TITL XADSUB
;
; PROGRAMM ZUR SEDEZIMALEN ADDITION UND SUBTRAKTION
; ZWEIER 16 BIT DUALZAHLEN MIT DEM SCMP
; ZEITBEDARF SUB = 300 MIKROSEKUNDEN
; ZEITBEDARF ADD = 310 MIKROSEKUNDEN
; BEI 2 MHZ TAKT
;
;
.LOC 0550
;
0550 08    XADSB:NOP              ;
0551 9C0F        JNZ SUB          ;SPRUNG, WENN AC NICHT 0
0553 02    ADD:  CCL              ;CY/L MUSS 0 SEIN
0554 C602        LD @02(2)        ; SUM1L => AC
0556 F602        ADD @02(2)       ;
0558 CA00        ST 00(2)         ;SUML = SUM1L + SUM2L
055A C2FD        LD FD(2)         ;SUM1H => AC
055C F2FF        ADD FF(2)        ;
055E CA01        ST 01(2)         ;SUMH = SUM1H + SUM2H + CY/L
0560 900D        JMP COND         ;ADDITION BEENDET
0562 03    SUB:  SCL              ;CY/L = 1 FUER SUBTRAKTION
0563 C602        LD @02(2)        ;MINL => AC
0565 FE02        CAD @02(2)       ;
0567 CA00        ST 00(2)         ;DIFFL = MINL - SUBL
0569 C2FD        LD FD(2)         ;MINH => AC
056B FAFF        CAD FF(2)        ;
056D CA01        ST 01(2)         ;DIFFH = MINH - SUBH
056F 06    COND: CSA              ;CY/L UND OV IM AC UEBERGEBEN
0570 I4C0        ANI C0           ;
0572 3F          XPPC3            ;RUECKSPRUNG
0573 90DB        JMP XADSB        ;WIEDEREINTRITTSPUNKT BEI MEHRFACHAUFRUF
;
;
.END
```

4.5 Dezimale Addition und Subtraktion mit dem Programm »DADSUB«

Ähnlich wie beim Programm in Kapitel 4.4 soll auch hier die Stacktechnik angewendet werden. Auch hier wird mit dem Akkumulatorinhalt gesteuert, ob addiert oder subtrahiert werden soll.
Die Rechenoperanden haben je sechs signifikante Stellen. Durch die interne BCD-Darstellung werden dafür je drei Byte benötigt. Das Ergebnis wird in vier Byte abgelegt. Das höchstwertige Byte des Resultats enthält in den oberen vier Bit das Vorzeichen. F_{16} bedeutet negatives, 0_{16} bedeutet positives Vorzeichen. Nach Ausführung des Programms zeigt der Stackpointer P2 auf die niederwertigste Stelle des Ergebnisses.
Die dezimale Subtraktion wird ebenso wie die sedezimale Subtraktion durch Addition des Komplements durchgeführt. Nur das wir hier das Komplement zur nächsthöheren Zehnerpotenz benutzen müssen, und das ist 100_{10}. Durch den Befehlsvorrat des SCMP bedingt wird die Subtraktion wie folgt ausgeführt.
Zunächst wird das Zweierkomplement des Subtrahenden gebildet. Subtrahiert man davon die Konstante 66 indem man ihr Zweierkomplement ($9A_{16}$) addiert, so erhält man das 100er-Komplement des Subtrahenden. Die Zahl 66 deshalb, weil die sedezimale Zahlendarstellung sechs Zeichen mehr als die Dezimaldarstellung kennt, und das gilt für jede BCD-Stelle.
Das Hunderterkomplement des Subtrahenden wird jetzt dezimal zum Minuenden addiert. Die Rechnung liefert uns

$$\text{Minuend}_{10} - \text{Subtrahend}_{10} = \text{Differenz}_{10}$$

Der Aufbau der Logik von »ADD« und »SUB« in DADSUB wurde unterschiedlich gewählt um zwei verschiedene Programmiertechniken zu veranschaulichen. Im Teil ADD wurde »geradeaus« programmiert und im Teil SUB wird die Berechnung in einer Schleife durchgeführt.

Stackbelegung

Stack	Addition		Subtraktion
+ 0;	Summand 1, Einer/Zehner-Stelle	;	Minuend, Einer/Zehner-Stelle
+ 1;	Summand 1, Hundert./Taus.-Stelle	;	Minuend, Hund./Taus.-Stelle
+ 2;	Summand 1, Zehntaus./H.Taus.-Stelle	;	Minuend, Zehnt./H.Taus.-Stelle
+ 3;	Summand 2, Einer/Zehner-Stelle	;	Subtrahend, Einer/Zehner-Stelle
+ 4;	Summand 2, Hundert./Taus.-Stelle	;	Subtrahend, Hund./Taus.-Stelle
+ 5;	Summand 2, Zehntaus./H.Taus.-Stelle	;	Subtrahend, Zehnt./H.Taus.-Stelle
+ 6;	Summe, Einer/Zehner-Stelle	;	Differenz, Einer/Zehner-Stelle
+ 7;	Summe, Hund./Tausender-Stelle	;	Differenz, Hund./Taus.-Stelle
+ 8;	Summe, Zehntaus./Hund.Taus.-Stelle	;	Differenz, Zehnt./H.Taus.-Stelle
+ 9;	Summe, Vorzeichen/Mill.-Stelle	;	Differenz, Vorz./Mill.-Stelle

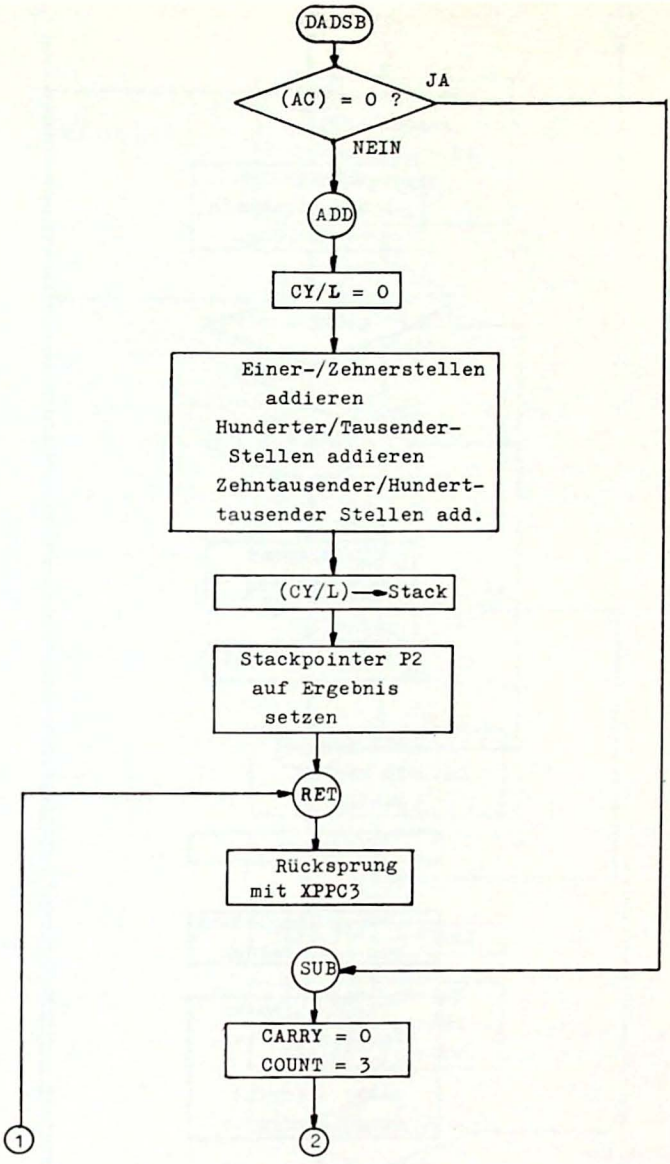


Bild 4.5.1: Flußdiagramm zum Programm »DADSUB«

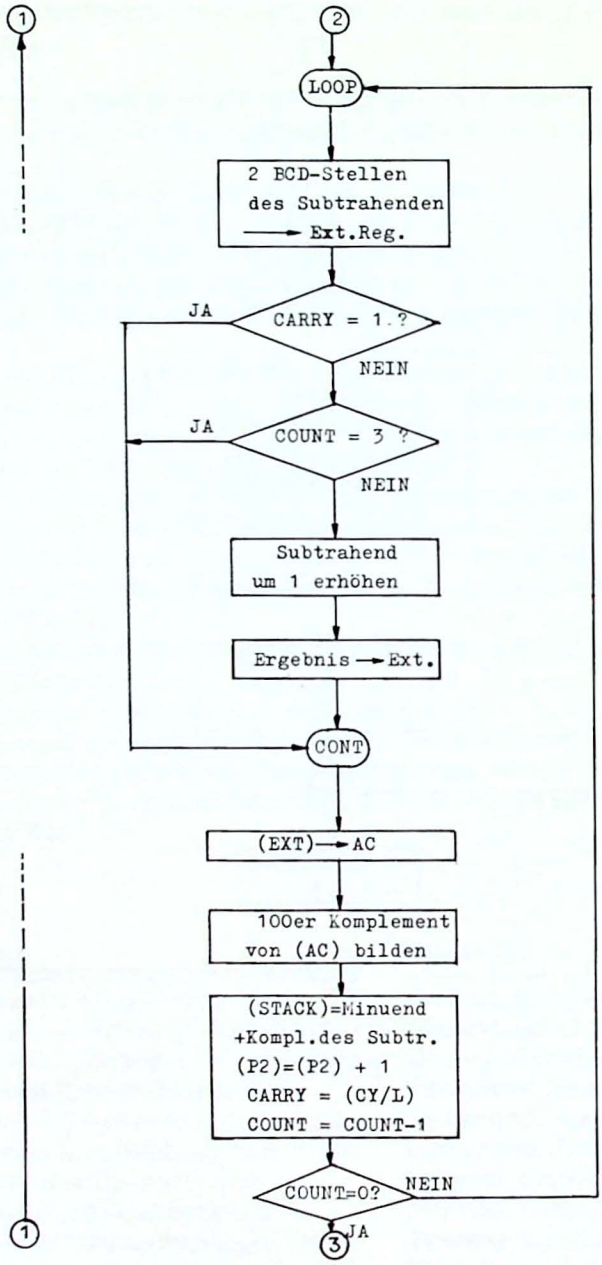


Bild 4.5.1: Fortsetzung

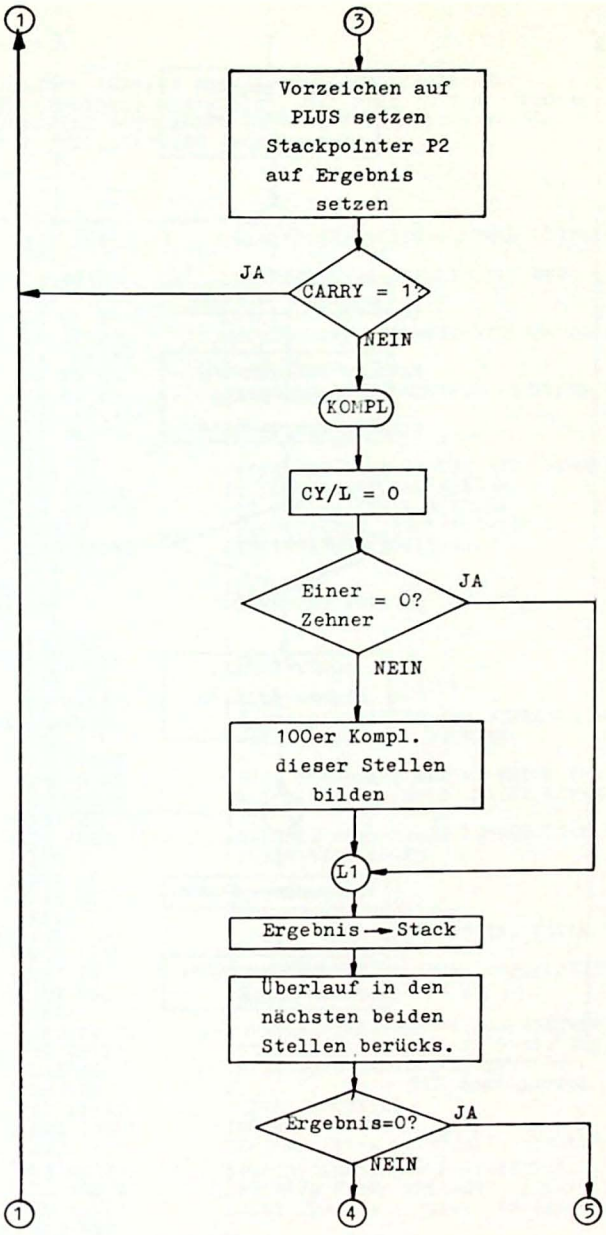


Bild 4.5.1: Fortsetzung

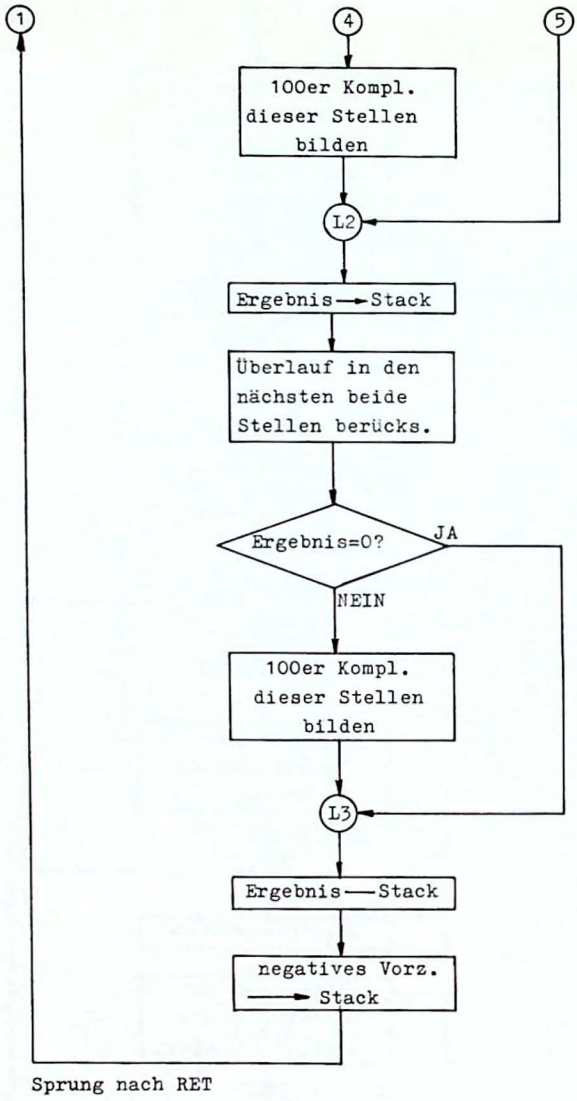


Bild 4.5.1: Fortsetzung

```
;
;
        .TITL DADSUB
;
; MIT DIESEM PROGRAMM KOENNEN ZWEI 6-STELLIGE BCD-ZAHLEN
; ADDIERT WERDEN (ERGEBNIS 7-STELLIG), BEI 2MHZ IN 518 MIKROSEK.
; BZW. ZWEI 6-STELLIGE BCD-ZAHLEN KOENNEN SUBTRAHIERT WERDEN,
; BEI 2MHZ IN 2,4 MILLISEKUNDEN (WORST CASE)
;
.LOC 0500
;
0500 08    DADSB:NOP              ;
0501 9C20         JNZ SUB         ;WENN AKKU-INHALT=1, DANN SUBTRAKTION
0503 02    ADD:   CCL             ;
0504 C200         LD 00(2)        ;EINER/ZEHNER-STELLEN ADDIEREN
0506 EA03         DAD 03(2)       ;
0508 CA06         ST 06(2)        ;ERGEBNIS SICHERN
050A C201         LD 01(2)        ;HUNDERTER/TAUSENDER-STELLEN ADDIEREN
050C EA04         DAD 04(2)       ;
050E CA07         ST 07(2)        ;ERGEBNIS SICHERN
0510 C202         LD 02(2)        ;ZEHNTAUS./HUNDERTTAUS.-STELLEN ADD.
0512 EA05         DAD 05(2)       ;
0514 CA08         ST 08(2)        ;ERGEBNIS SICHERN
0516 C400         LDI 00          ;
0518 EC00         DAI 00          ;WERT DES CY/L IN DEN AKKU BRINGEN
051A CA09         ST 09(2)        ;MILLIONEN-STELLE BILDEN
051C C606         LD @06(2)       ;P2 AUF ERGEBNIS STELLEN
051E 3F    RET:   XPPC3           ;RUECKSPRUNG UEBER POINTER 3
051F 901F         JMP DADSB       ;WIEDEREINTRITTSADRESSE
;
0521 00    COUNT:00               ;ZAEHLER
0522 00    CARRY:00               ;SPEICHER FUER CY/L-STATUS
;
0523 C403 SUB:    LDI 03          ;
0525 C8FB         ST COUNT        ;COUNT = 3
0527 C400         LDI 00          ;
0529 C8F8         ST CARRY        ;CARRY = 0
052B C203 LOOP:   LD 03(2)        ;ZWEI BCD-STELLEN DES SUBTRAH. LADEN
052D 01           XAE             ;UND IM EXT.REG. SICHERN
052E C0F3         LD CARRY        ;
0530 9C0F         JNZ CONT        ;WENN UEBERLAUF WAR => KEINE KORR. V.SUBTRA
0532 C0EE         LD COUNT        ;BEI EINER/ZEHNERST. KEINE KORREKTUR
0534 E403         XRI 03          ;
0536 9805         JZ CONT         ;SPRUNG, WENN EINER/ZEHNER-STELLE
0538 40           LDE             ;SUBTRAHEND HOLEN
0539 02           CCL             ;
053A EC01         DAI 01          ;SUBTRAH.+1
053C 01           XAE             ;IN EXT.REG. SICHERN
053D 40    CONT:  LDE             ;SUBTR. ODER KORR. SUBTR. LADEN
053E 03           SCL             ;
053F E4FF         XRI FF          ;HUNDERTER-KOMPLEMENT DES SUBTRAH. BILDEN
0541 F49A         ADI 9A          ;KONSTANTE ADDIEREN
0543 02           CCL             ;
0544 EE01         DAD @01(2)      ;MINUEND+100ER-KOMPL.DES SUBTRAHENDEN
0546 CA05         ST 05(2)        ;ERGEBNIS IN STACK, P2 WURDE INKREMENTIERT
0548 C400         LDI 00          ;CY/L NACH CARRY BRINGEN
054A EC00         DAI 00          ;
054C C8D5         ST CARRY        ;CARRY = <CY/L>
054E B8D2         DLD COUNT       ;COUNT = COUNT-1
0550 9C19         JNZ LOOP        ;SPRUNG, WENN NICHT ALLE STELLEN SUBTRAHIERT
0552 CA06         ST 06(2)        ;VORZEICHENBYTE = 0 (POSITIV)
0554 C603         LD @03(2)       ;POINTER 2 AUF ERGEBNIS SETZEN
0556 C0CB         LD CARRY        ;WENN CARRY = 1, DANN IST DAS ERGEBNIS POSITI
0558 9CC4         JNZ RET         ;
055A 02    KOMPL:CCL              ;KOMPLEMENT ZU EINER MILLION BILDEN
055B C200         LD 00(2)        ;EINER/ZEHNER
055D 9805         JZ L1           ;
```

```
055F 03          SCL             ;
0560 E4FF        XRI FF          ;
0562 F49A        ADI 9A          ;
0564 CA00 L1:    ST 00(2)        ;ERGEBNIS EINER/ZEHNER-STELLE
0566 C201        LD 01(2)        ;HUNDERTER/TAUSENDER
0568 EC00        DAI 00          ;
056A 9805        JZ L2           ;
056C 03          SCL             ;
056D E4FF        XRI FF          ;
056F F49A        ADI 9A          ;
0571 CA01 L2:    ST 01(2)        ;ERGEBNIS HUNDERTER/TAUSENDER-STELLE
0573 C202        LD 02(2)        ;ZEHNTAUSENDER/HUNDERTTAUSENDER
0575 EC00        DAI 00          ;
0577 9805        JZ L3           ;
0579 03          SCL             ;
057A E4FF        XRI FF          ;
057C F49A        ADI 9A          ;
057E CA02 L3:    ST 02(2)        ;ERGEBNIS ZEHNTAUS./HUNDERTTAUS.-STELLE
0580 C4F0        LDI F0          ;
0582 CA03        ST 03(2)        ;VORZEICHENBYTE = F0 (NEGATIV)
0584 9097        JMP RET         ;RUECKSPRUNG
;
;
.END
```

4.6 Hex/Dezimal- und Dezimal/Hex-Wandlung mit dem Programm »DEZHEX«

Mit dem Programm »DEZHEX« ist es möglich eine 24 Bit lange Dualzahl (negative Zahlen im Zweierkomplement) in eine vorzeichenbehaftete Dezimalzahl in BCD-Darstellung umzuwandeln. Das Ergebnis besteht in diesem Fall aus fünf Stellen plus eine Stelle für das Vorzeichen.

Umgekehrt kann eine Dezimalzahl im Bereich ± 99999 in eine entsprechende Binärzahl umgewandelt werden.

Der Bereich der Dualzahl reicht demnach von FE7961 bis 1869F. Der Programmaufruf erfolgt über Pointer 3. Der Inhalt des Akkumulators gibt den Umwandlungsmodus an.

AC = 0 bedeutet Hex/Dezimal-Wandlung, AC = 1 bedeutet Dezimal/Hex-Wandlung. Pointer 2 zeigt auf die höchstwertige Stelle der übergebenen Zahl. Es sind drei Byte als Übergabespeicherbereich festgelegt.

Belegung des durch Pointer 2 adressierten Übergabebereichs.

Adresse	Dez. → Hex. nach Aufruf Hex. → Dez. vor Aufruf	Dez. → Hex. vor Aufruf Hex. → Dez. nach Aufruf
(P2) + 0	; höherwertiges Byte	; Vorzeichen und Zehntausender-Stelle
(P2) + 1	; mittleres Byte	; Hunderter und Tausender-Stelle
(P2) + 2	; niederwertiges Byte	; Zehner und Einer-Stelle

Positives Vorzeichen wird durch eine »0« in den oberen vier Bit des höchstwertigen Byte des Übergabebereichs dargestellt. Bei negativem Vorzeichen sind diese vier Bit »1« (F_{16}).
Durch geringfügige Änderungen kann das Umwandlungsprogramm auf fast beliebige Wortbreite erweitert werden.

Erläuterungen zur Dezimal-/Sedezimalwandlung:
Hier wird das bereits in Kapitel 1.3 beschriebene Verfahren der fortgesetzten Division durch zwei angewendet. Die Reste der Divisionen ergeben das Ergebnis in Dualdarstellung. Die Division durch zwei wird durch rechtsschieben um eine Stelle durchgeführt. Da es sich jedoch hier um die Division einer BCD-Zahl handelt muß das Resultat korrigiert werden. Immer, wenn bei der Division eine »1« in die nächste Dekade geschoben wird, muß diese um drei vermindert werden. Deutlich wird das, wenn man die Zahl 70 in BCD-Darstellung »von Hand« durch zwei teilt.
Ist die umzuwandelnde Zahl negativ, so muß nach der Umwandlung das Zweierkomplement des Ergebnisses gebildet werden.

Erläuterungen zur Sedezimal-/Dezimalwandlung:
In diesem Programmteil wird einfach die Logik von »DZHX« umgekehrt. Aus der allgemeinen Form der Koeffizientenschreibweise einer

Dualzahl $B_{23} \cdot 2^{23} + B_{22} \cdot 2^{22} + B_{21} \cdot 2^{21} + \ldots B_2 \cdot 2^2 + B_1 \cdot 2^1 + B_0 \cdot 2^0$

wird durch Umstellung

$$(\ldots((B_{23} \cdot 2 + B_{22}) \cdot 2 + B_{21}) \cdot 2 \ldots + B_2) \cdot 2 + B_1) \cdot 2 + B_0$$

Vor der eigentlichen Umwandlung werden jedoch negative Dualzahlen erst positiv gemacht. Das negative Vorzeichen (F_{16}) wird danach der Dezimalzahl wieder zugeführt. Eine Überprüfung der eingegebenen Dualzahlen auf eine eventuelle Bereichsüberschreitung wird nicht vorgenommen.
Ein Teil der Programmlogik ist für die Umwandlung selbst überflüssig, wurde jedoch eingebaut um das Programm leichter auf mehr Stellen erweitern zu können.

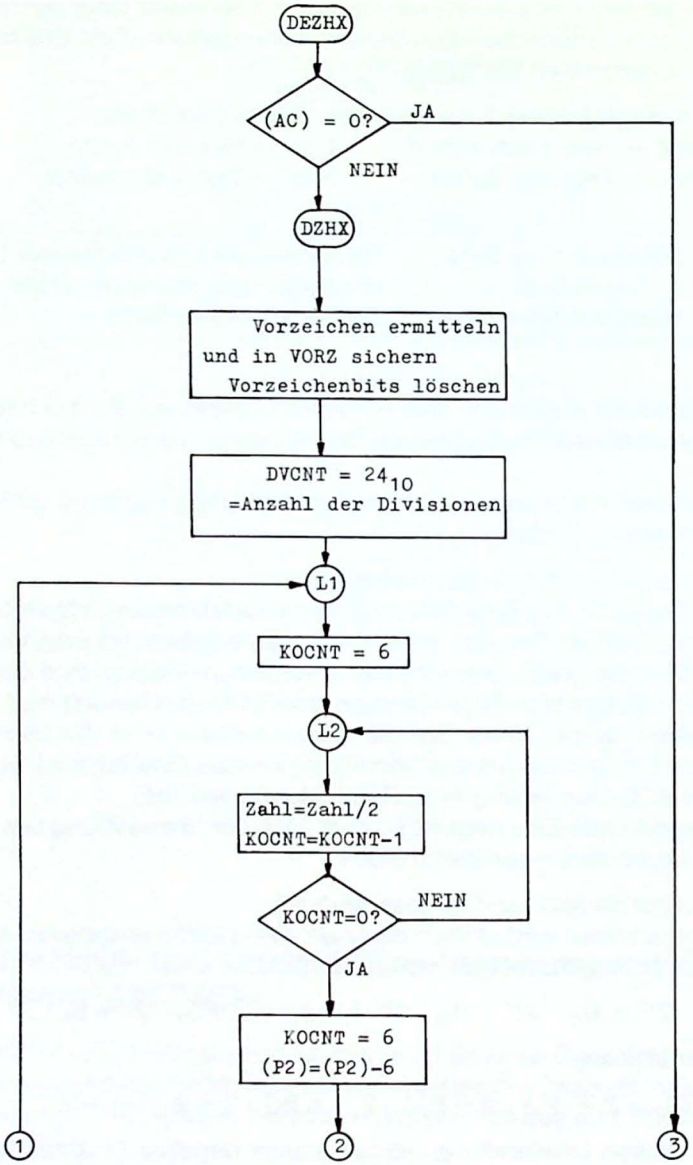


Bild 4.6.1: Flußdiagramm zum Programm »DEZHEX«

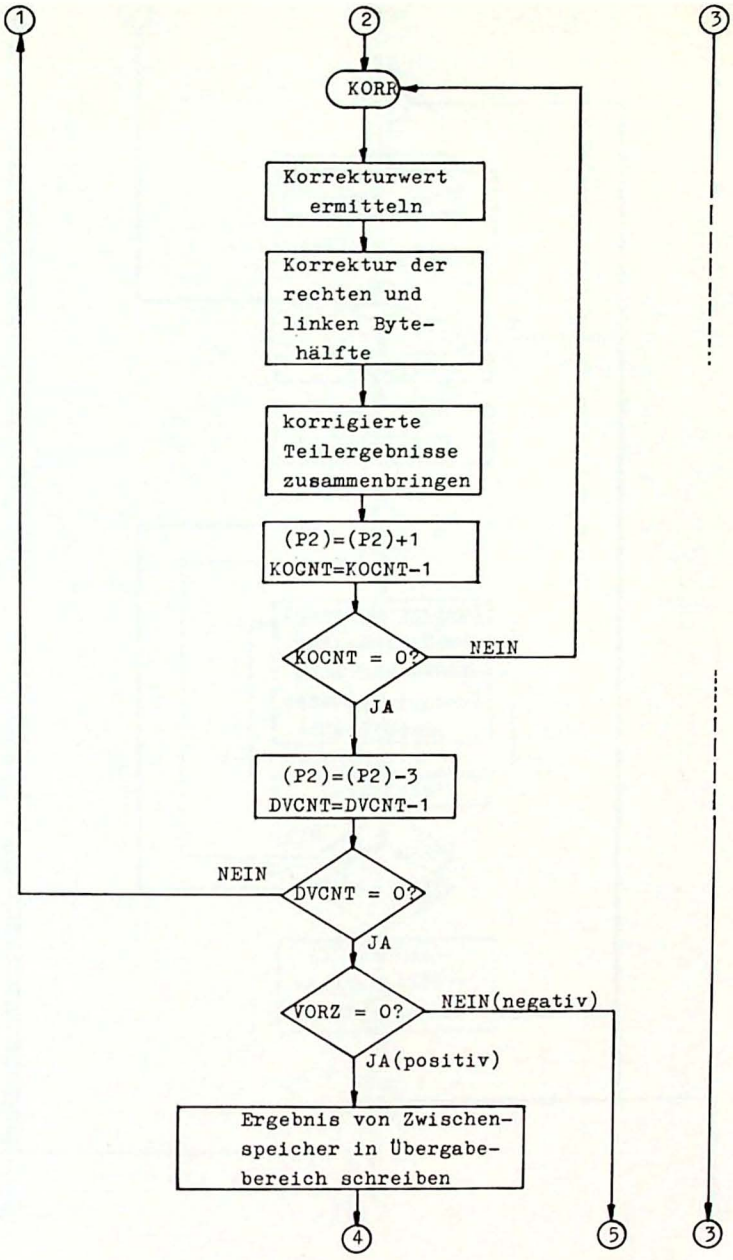


Bild 4.6.1: Fortsetzung

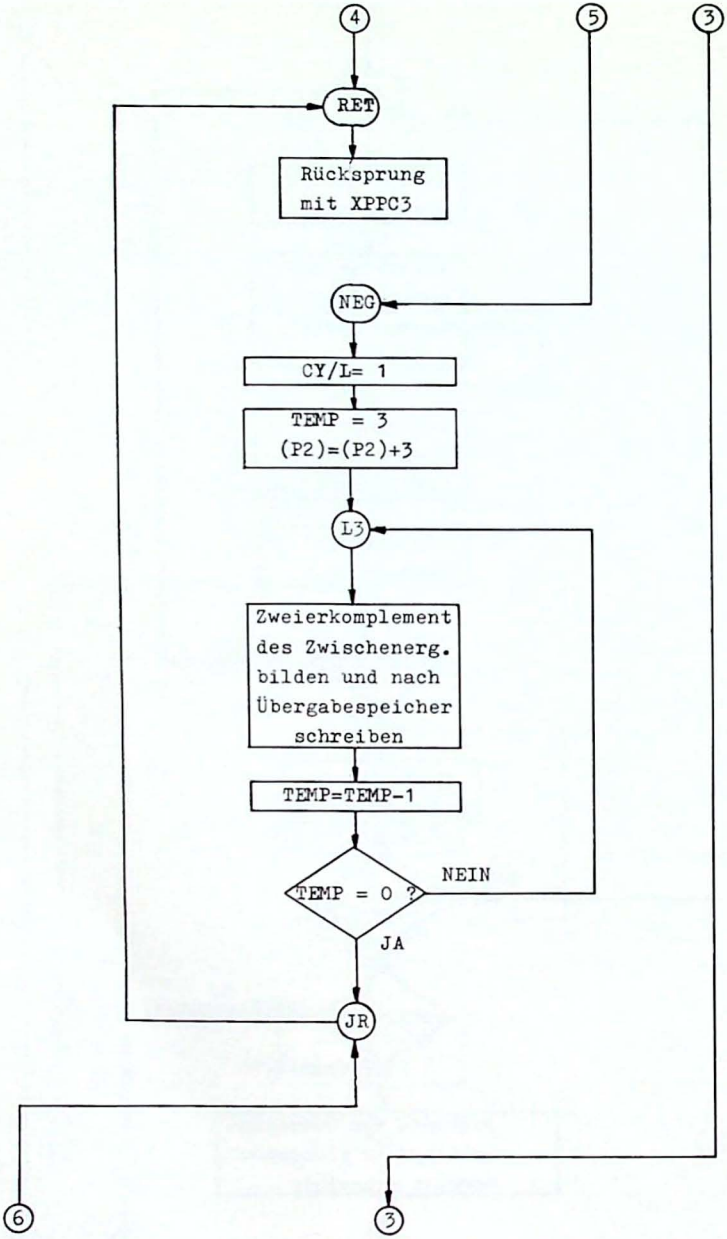


Bild 4.6.1: Fortsetzung

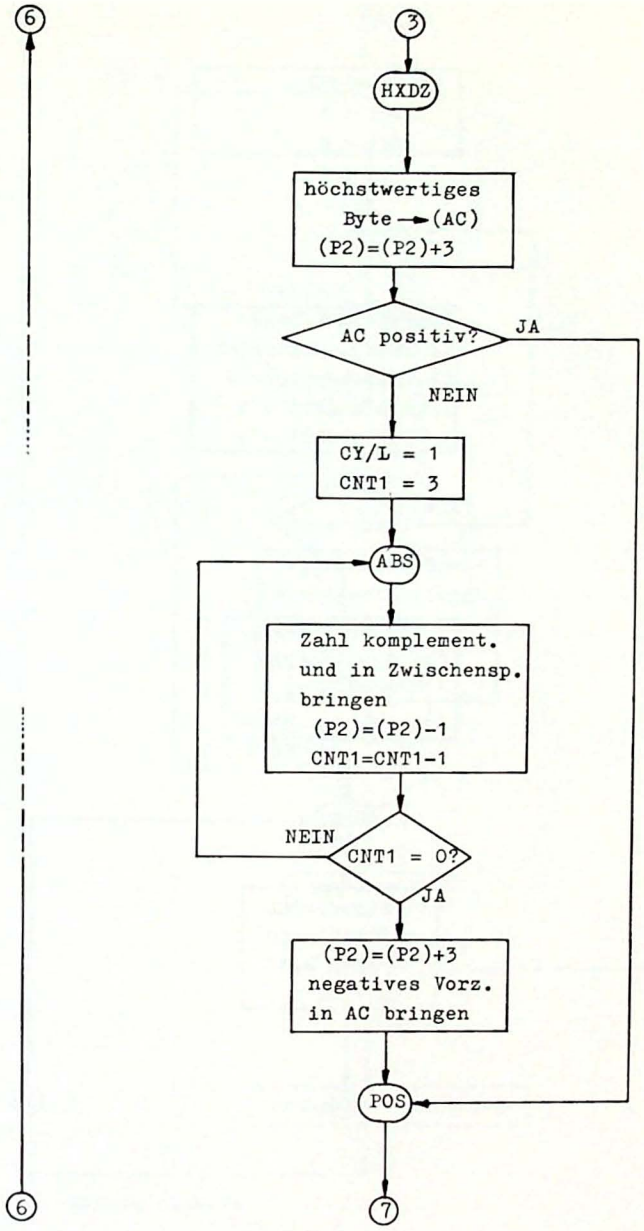


Bild 4.6.1: Fortsetzung

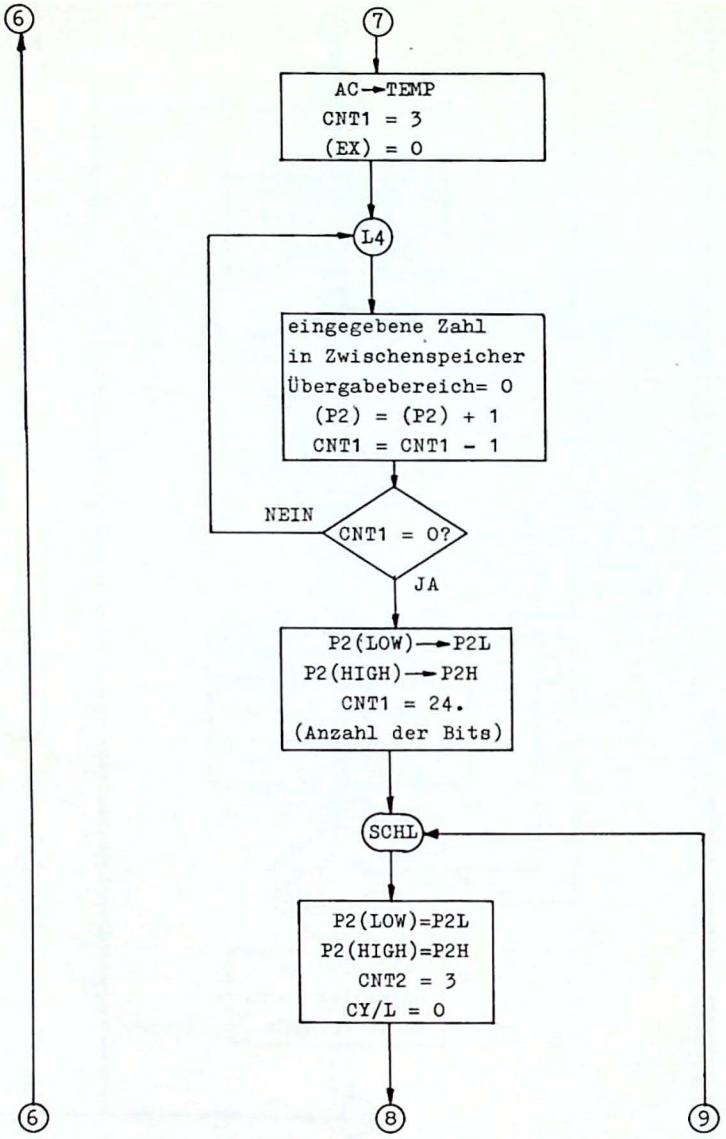


Bild 4.6.1: Fortsetzung

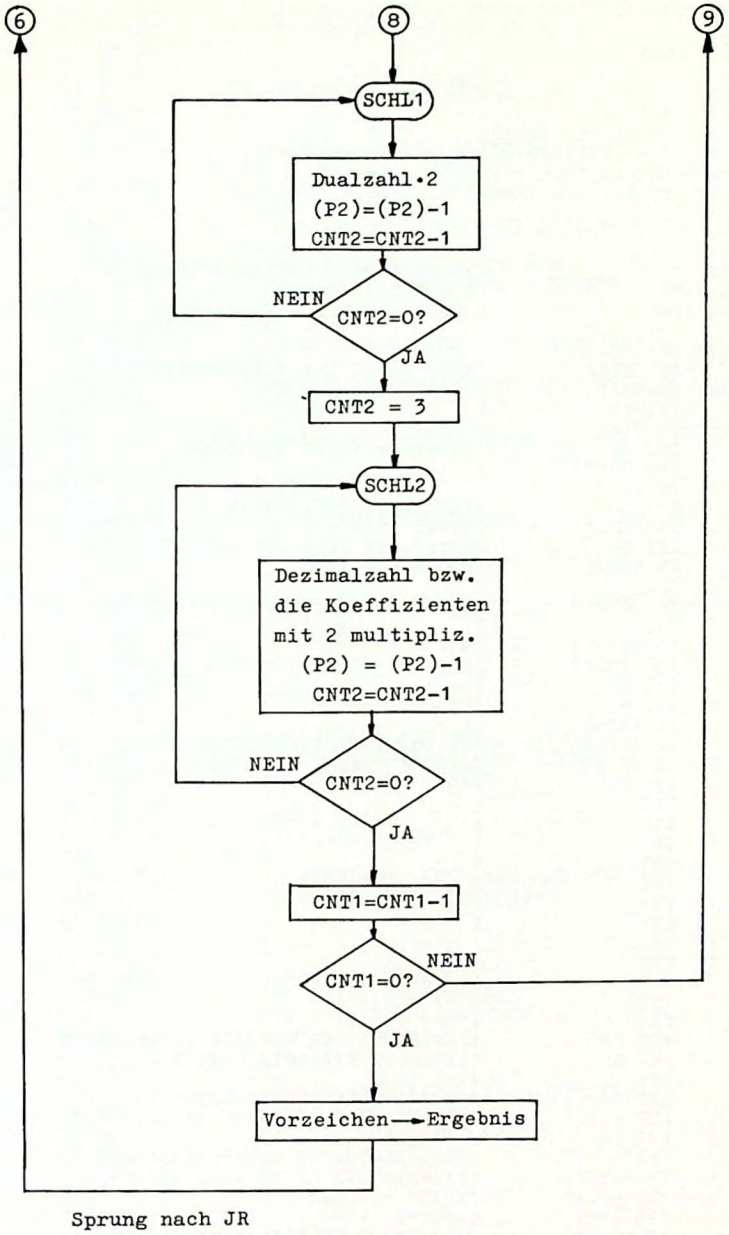


Bild 4.6.1: Fortsetzung

```
;
;
        .TITL DEZHEX
;
; UMWANDLUNGSPROGRAMM DEZIMAL ( VORZ. U. 5 STELLEN BCD)
; IN SEDEZIMAL (20 BIT)
; UND SEDEZIMAL IN DEZIMAL (BCD)
; ZEITBEDARF IN MILLISEKUNDEN BEI 2 MHZ TAKTFREQUENZ
; DEZ-/HEX-WANDLUNG CA. 57,4 (POSITIV), 58 (NEGATIV)
; HEX-/DEZ-WANDLUNG CA. 102 (POSITIV), 102,6 (NEGATIV)
;
.LOC 042D
;
042D 00   VORZ: 0               ;VORZEICHEN ZWISCHENSPEICHER
042E 00   DVCNT:00              ;DIVISIONSZAEHLER
042F 00   KOCNT:00              ;KORREKTUR SCHLEIFENZAEHLER
;
0430 08   IEZHX:NOP            ;
0431 985F       JZ JHXDZ        ;WENN AC=0, HEX-/DEZ.-WANDLUNG
0433 C200 DZHX: LD 00(2)        ;VORZEICHEN SICHERN
0435 D4F0       ANI F0          ;
0437 C8F5       ST VORZ         ;
0439 C40F       LDI 0F          ;VORZEICHENBITS LOESCHEN
043B D200       AND 00(2)       ;
043D CA00       ST 00(2)        ;
043F C418       LDI 18          ;KONSTANTE (24.) FUER 3 BYTE LADEN
0441 C8EC       ST DVCNT        ;DVCNT=18
0443 C406 L1:   LDI 06          ;KONSTANTE FUER 6 BCD-DIGITS
0445 C8E9       ST KOCNT        ;KOCNT=6
0447 02         CCL             ;
0448 C200 L2:   LD 00(2)        ;BCD-WERT DURCH ZWEI DIVIDIEREN
044A 1F         RRL             ;
044B CE01       ST @01(2)       ;
044D B8E1       DLD KOCNT       ;KOCNT = KOCNT-1
044F 9CF7       JNZ L2          ;
0451 C403       LDI 03          ;KOCNT = 3
0453 C8DB       ST KOCNT        ;
0455 C6FA       LD @FA(2)       ;P2 UM 6 ZURUECK SETZEN
0457 C200 KORR: LD 00(2)        ;KORREKTURWERT ERMITTELN
0459 D488       ANI 88          ;
045B 01         XAE             ;
045C 40         LDE             ;
045D 1C         SR              ;
045E 58         ORE             ;
045F 01         XAE             ;
0460 1C         SR              ;
0461 1C         SR              ;
0462 1C         SR              ;
0463 58         ORE             ;
0464 01         XAE             ;
0465 40         LDE             ;
0466 02         CCL             ;
0467 F200       ADD 00(2)       ;KORREKTUR DER RECHTEN BYTEHAELFTE
0469 D40F       ANI 0F          ;LINKE BYTEHAELFTE LOESCHEN
046B 01         XAE             ;
046C D4F0       ANI F0          ;RECHTE BYTEHAELFTE LOESCHEN
046E F200       ADD 00(2)       ;KORREKTUR DER LINKEN BYTEHAELFTE
0470 D4F0       ANI F0          ;
0472 58         ORE             ;TEILERGEBNISSE ZUSAMMENBRINGEN
0473 CE01       ST @01(2)       ;SICHERN UND P2 UM EINS ERHOEHEN
0475 B8B9       DLD KOCNT       ;KOCNT = KOCNT-1
0477 9CDE       JNZ KORR        ;SPRUNG, WENN KORREKTUR NICHT FERTIG
0479 C6FD       LD @FD(2)       ;POINTER 2 UM 3 ZURUECKSETZTEN
047B B8B2       DLD DVCNT       ;DVCNT = DVCNT-1
047D 9CC4       JNZ L1          ;SPRUNG, WENN NICHT ALLE DIVISIONEN FERTIG
047F C0AD       LD VORZ         ;
0481 9C12       JNZ NEG         ;SPRUNG, WENN NEGATIVES VORZEICHEN
```

```
04E3 C203       LD 03(2)        ;ERGEBNIS AUS ZWISCHENSPEICHER IN
0485 CA00       ST 00(2)        ;UEBERGABESPEICHER BRINGEN
04E7 C204       LD 04(2)        ;
0489 CA01       ST 01(2)        ;
04EB C205       LD 05(2)        ;
04ED CA02       ST 02(2)        ;
04EF 3F   RET:  XPPC3           ;RUECKSPRUNG
0490 909D       JMP DEZHX       ;WIEDEREINTRITTSADRESSE
;
0492 9018 JHXDZ:JMP HXDZ        ;SPRUNGVERLAENGERUNG
0494 00   TEMP: 00              ;HILFSSPEICHER
0495 03   NEG:  SCL             ;ZWEIERKOMPLEMENT DES ERGEBNISSES BILDEN
0496 C403       LDI 03          ;
0498 C8FB       ST TEMP         ;SCHLEIFENZAEHLER TEMP=3
049A C603       LD @03(2)       ;P2 AUF ZWISCHENERG.+1 SETZEN
049C C400 L3:   LDI 00          ;
049E FA02       CAD 02(2)       ;
04A0 C8FF       ST @FF(2)       ;ERGEBNIS SICHERN, P2 DEKREMENTIEREN
04A2 BBF1       DLD TEMP        ;
04A4 9CF6       JNZ L3          ;SPRUNG SOLANGE NICHT ALLE STELLEN FERTIG
04A6 90E7 JR:   JMP RET         ;RUECKSPRUNG
;
04A8 00   P2L:  00              ;ZWISCHENSPEICHER FUER P2(LOW)
04A9 00   P2H:  00              ;                      P2(HIGH)
04AA 00   CNT1: 00              ;ZAEHLER 1
04AB 00   CNT2: 00              ;ZAEHLER 2
04AC C603 HXDZ: LD @03(2)       ;MSB LADEN UND P2 AUF LSB+1 SETZEN
04AE D480       ANI 80          ;
04B0 9813       JZ POS          ;SPRUNG, WENN POSITIVE DUALZAHL
04B2 03         SCL             ;
04B3 C403       LDI 03          ;SCHLEIFENZAEHLER FUER ABS.WERT SETZEN
04B5 C8F4       ST TEMP         ;TEMP=3
04B7 C400 ABS:  LDI 00          ;ABSOLUTWERT BILDEN
04B9 FFFF       CAD @FF(2)      ;
04BF CA00       ST 00(2)        ;
04BD B8EC       DLD CNT1        ;CNT1=CNT1-1
04BF 9CF6       JNZ ABS         ;
04C1 C603       LD @03(2)       ;POINTER 2 AUF LSB+1 SETZEN
04C3 C4F0       LDI F0          ;NEGATIVES VORZEICHEN BILDEN
04C5 C8CE POS:  ST TEMP         ;VORZEICHEN SICHERN
04C7 C403       LDI 03          ;
04C9 C8E0       ST CNT1         ;CNT1 = 3
04CB C400       LDI 00          ;EXT.REG. = 00
04CD 01         XAE             ;
04CF C2FD L4:   LD FD(2)        ;EINGABEWERT NACH ZWISCHENSPEICHER BRINGEN
04D0 01         XAE             ;EINGABEPUFFER LOESCHEN
04D1 CAFD       ST FD(2)        ;
04D3 01         XAE             ;
04D4 CE01       ST @01(2)       ;
04D6 B8D3       DLD CNT1        ;CNT1 = CNT1-1
04D8 9CF4       JNZ L4          ;SCHLEIFE BIS FERTIG
04DA 32         XPAL2           ;P2 SICHERN, STEHT AUF ZW.SP. LSB+1
04DB C8CC       ST P2L          ;
04DD 36         XPAH2           ;
04DE C8CA       ST P2H          ;
04E0 C418       LDI 18          ;ANZAHL DER BITS DER DUALZAHL (24.)
04E2 C8C7       ST CNT1         ;CNT1 = 18
04E4 C0C4 SCHL: LD P2H          ;P2 AUF STARTWERT SETZEN
04E6 36         XPAH2           ;
04E7 C0C0       LD P2L          ;
04E9 32         XPAL2           ;
04EA C403       LDI 03          ;
04EC C8BE       ST CNT2         ;CNT2 = 3
04EE 02         CCL             ;
04EF C2FF SCHL1:LD FF(2)        ;DUALZAHL MIT 2 MULTIPLIZIEREN
04F1 F2FF       ADD FF(2)       ;
04F3 CEFF       ST @FF(2)       ;P2 DEKREMENTIEREN UND ERG. SICHERN
```

```
04F5 B8B5       DLD CNT2        ;CNT2 = CNT2-1
04F7 9CF6       JNZ SCHL1       ;SCHLEIFE BIS ALLE BYTE MULT.
04F9 C403       LDI 03          ;
04FB C8AF       ST CNT2         ;CNT2 = 3, DREI GRUPPEN A 2 DEK. BCD-ZAHL
04FD C2FF SCHL2:LD FF(2)        ;DEZ.ZAHL (KOEFFIZIENTEN) MIT 2 MULT.
04FF FAFF       DAD FF(2)       ;
0501 C8FF       ST @FF(2)       ;
0503 B8A7       DLD CNT2        ;
0505 9CF6       JNZ SCHL2       ;SCHLEIFE
0507 B8A9       DLD CNT1        ;
0509 9CD9       JNZ SCHL        ;AUSSENSCHLEIFE
050B C0F8       LD TEMP         ;UMWANDLUNG BEENDET
050D DA00       OR 00(2)        ;VORZEICHEN IN ERGEBNIS BRINGEN
050F CA00       ST 00(2)        ;
0511 9092       JMP JR          ;RUECKSPRUNG
;
;
.END
```

4.7 SCMP als Tonfrequenzgenerator mit dem Programm »TONFREQ«

Im Beispielprogramm soll die Möglichkeit des SCMP direkt mit der Außenwelt zu verkehren ausgenutzt werden. Mit den Flags 0, 1 und 2 soll eine Rechteckschwingung im Hörbereich erzeugt werden. Die Frequenzverhältnisse der an den Flags entstehenden Schwingungen F0:F1:F2 sind 1:0,5:0,25.

Mit Betätigen von SENSE A nimmt die Frequenz ab, mit SENSE B zu. Die untere Grenzfrequenz ist 11 Hz und die höchstmögliche Frequenz ist 1588 Hz.

Die Frequenz kann umlaufend eingestellt werden, d.h., überschreitet man die obere Grenze, so kommt man gleich an die untere Grenze. Bei Betätigung von SENSE A tritt der umgekehrte Effekt auf. In diesem Programm macht sich schon störend bemerkbar, daß der SCMP doch recht langsam ist. Das macht diesen Mikroprozessor, wie bereits gesagt, nicht generell unbrauchbar.

TONFREQ soll in erster Linie als Demonstrationsprogramm dienen. Mit nachfolgenden Befehlen kann bei einer Taktfrequenz von 2 MHz eine Rechteckschwingung von 11380 Hz erzeugt werden; mehr geht nicht.

```
      LDI AA      ; Akku mit Bitmuster 10101010 laden
LOOP: CAS         ; Statusregister setzen
      RR          ; Bitmuster um eine Stelle nach rechts rotieren
      JMP LOOP    ; neues Bitmuster ausgeben
```

Durch unterschiedliche Bitmuster und Laufzeiten lassen sich auf diese Weise fast beliebige Signalfolgen realisieren.

Doch wieder zurück zu unserem Programm.

Die Zählervariable ist 16 Bit breit. Das entspricht einem Wertebereich von 0000_{16} bis $FFFF_{16}$. Alle Variablen werden im Programm über den Pointer 2 adressiert.

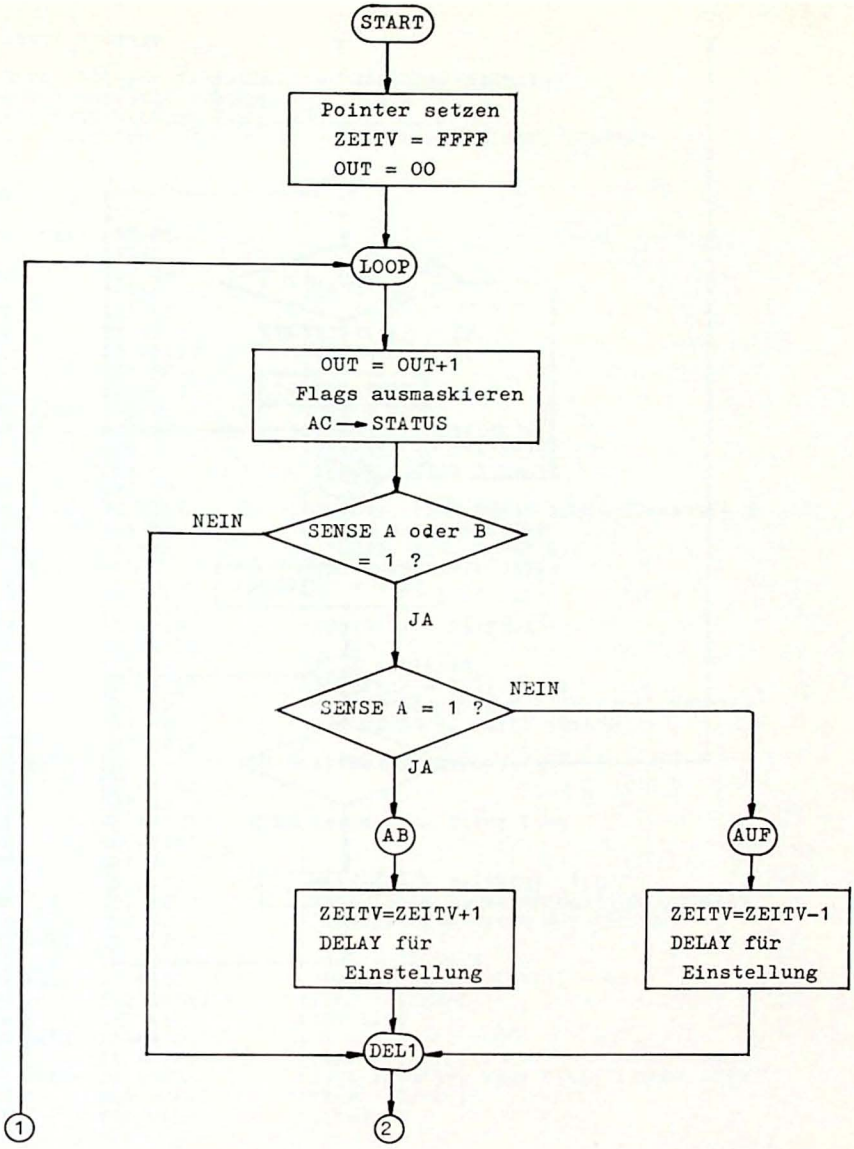


Bild 4.7.1: Flußdiagramm zum Programm »TONFREQ«

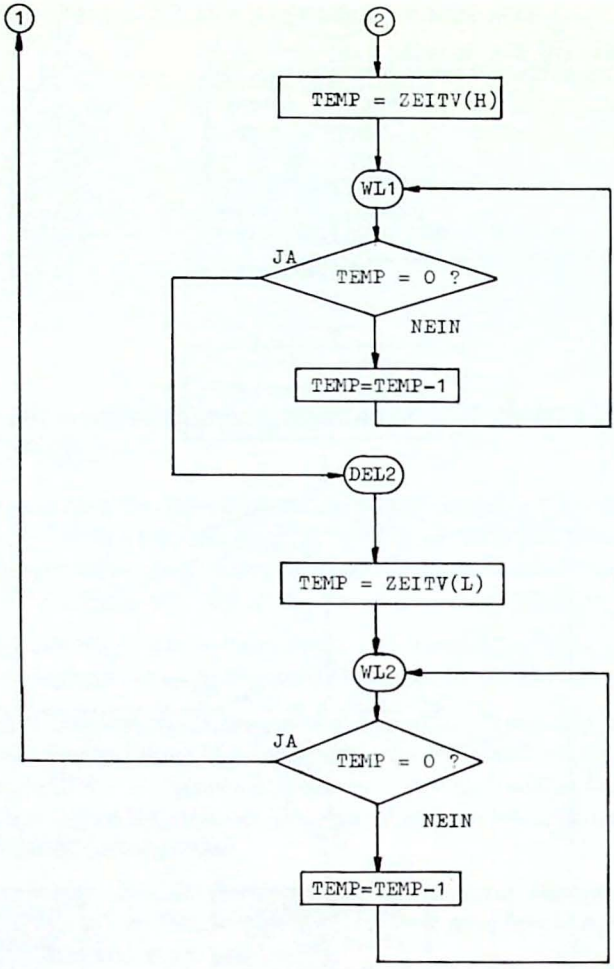


Bild 4.7.1: Fortsetzung

```
;
;
        .TITL TONFREQ
;
; MIT DIESEM PROGRAMM KANN DER SCMP ALS TONFREQUENZ-
; GENERATOR EINGESETZT WERDEN
; BEI EINER TAKTFREQUENZ VON 2 MHZ
; KOENNEN FREQUENZEN ZWISCHEN 11 UND 1588 HZ ERZEUGT WERDEN
;
;
.LOC 0420
;
0420 C405 START:LDI 05          ;
0422 36         XPAH2           ;
0423 C400       LDI 00          ;
0425 32         XPAL2           ;P2 = 500
0426 C4FF       LDI FF          ;
0428 CA00       ST 00(2)        ;ZEITV(L) = FF
042A CA01       ST 01(2)        ;ZEITV(H) = FF
042C C400       LDI 00          ;
042E CA03       ST 03(2)        ;OUT = 0
0430 AA03 LOOP: ILD 03(2)       ;OUT = OUT+1
0432 D407       ANI 07          ;FLAGS AUSMASKIEREN
0434 07         CAS             ;AUSGABE AN STATUSREGISTER
0435 06         CSA             ;SENSE A ODER B GESETZT ?
0436 D430       ANI 30          ;
0438 9828       JZ DEL1         ;SPRUNG, WENN BEIDE SENSE-EINGAENGE 0
043A D410       ANI 10          ;SENSE A AUSMASKIEREN
043C 9813       JZ AUF          ;SPRUNG, WENN B GESETZT
043E 02   AB:   CCL             ;ZEITV INKREMENTIEREN
043F C200       LD 00(2)        ;
0441 F401       ADI 01          ;
0443 CA00       ST 00(2)        ;ZEITV(L) = ZEITV(L)+1
0445 C201       LD 01(2)        ;
0447 F400       ADI 00          ;CY/L ADDIEREN
0449 CA01       ST 01(2)        ;ZEITV(H) = ZEITV(H)+1
044B C400       LDI 00          ;AENDERUNGSGESCHWINDIGKEIT EINSTELLEN
044D 8F80       DLY 80          ;VERZOEGERUNG (NACH WUNSCH)
044F 9011       JMP DEL1        ;
0451 03   AUF:  SCL             ;ZEITV DEKREMENTIEREN
0452 C200       LD 00(2)        ;
0454 FC01       CAI 01          ;
0456 CA00       ST 00(2)        ;ZEITV(L) = ZEITV(L)-1
0458 C201       LD 01(2)        ;
045A FC00       CAI 00          ;
045C CA01       ST 01(2)        ;ZEITV(H) = ZEITV(H)-CY/L
045E C400       LDI 00          ;AENDERUNGSGESCHWINDIGKEIT EINSTELLEN
0460 8F80       DLY 80          ;VERZOEGERUNG (NACH WUNSCH)
0462 C201 DEL1: LD 01(2)        ;
0464 CA02       ST 02(2)        ;TEMP = ZEITV(H)
0466 9804 WL1:  JZ DEL2         ;SPRUNG, WENN ZEITV(H) = 0
0468 BA02       DLD 02(2)       ;TEMP = TEMP-1
046A 90FA       JMP WL1         ;SCHLEIFE
046C C200 DEL2: LD 00(2)        ;
046E CA02       ST 02(2)        ;TEMP = ZEITV(L)
0470 98BE WL2:  JZ LOOP         ;NEUE AUSGABE, WENN VERZOEGERUNG BEENDET
0472 BA02       DLD 02(2)       ;TEMP = TEMP-1
0474 90FA       JMP WL2         ;SCHLEIFE
;
;
.END
```

Variablenliste:

Adresse	500_{16}	;	ZEITV(L)
	501_{16}	;	ZEITV(H)
	502_{16}	;	TEMP
	503_{16}	;	OUT

Einen größeren Tonumfang mit jeweils fest definierten Frequenzen kann man erreichen, indem z.B. bestimmten betätigten Tasten Werte zugeordnet werden. Diese Werte sind dann die frequenzbestimmenden Elemente

5. Der Mikroprozessor 6502 von Rockwell

Bei diesem Prozessor handelt es sich um einen der meist eingesetzten Mikroprozessoren auf dem Hobby-Computer bzw. Personalcomputer Markt. Er wird unter anderem in den Systemen PET, AIM, SYM und KIM eingesetzt. Bei einer Taktfrequenz von 1 MHz liegt die Ausführungszeit der Befehle zwischen 2 und 7 Mikrosekunden, d. h. man kann den 6502 zu den schnelleren Typen zählen, zumal auch eine 2 MHz Version angeboten wird. Der Adressierbereich ist 64KByte. Als besonderer Vorteil kann der sehr wirkungsvolle und durchsichtige Befehlssatz genannt werden.

Zusammenfassung der wichtigsten Daten des 6502:

16 Bit Adreßbus (40 Pin-Gehäuse) bzw. 12 Bit (13) Adreßbus im 28 Pin-Gehäuse.
8 Bit bidirektionaler Datenbus
nur eine Versorgungsspannung von 5V
1 MHz und 2 MHz Ausführung
DMA (direct memory access) fähig
Interrupteingänge für einen nicht maskierbaren Interrupt (NMI) und einen maskierbaren Interrupt (IRQ)
Buskompatibel zum Motorola MC6800; alle Motorola Peripherie-Bausteine der 6800-Generation sind anschließbar.
Der Stack kann einen variablen RAM-Bereich (Länge) haben, die Stackverwaltung ist sehr flexibel.
Ausführungen mit integriertem Taktgenerator sind erhältlich.
56 verschiedene Befehle mit einer Ausführungszeit von 2 bis 7 µs.
13 unterschiedliche Adressierungsarten.
Dezimal- und Binärarithmetik-Mode über Statusbit umschaltbar, wobei im Dezimalmode keinerlei Korrekturbefehle erforderlich sind.

Ein nicht zu vernachlässigender Vorteil des 6502 ist die relativ große Menge an Software die auf dem freien Markt angeboten wird.

Es stehen beim 6502 sechs bzw. sieben Adressierungsarten zur Verfügung, die in Kombination bis zu 13 unterschiedliche Adressierungsmöglichkeiten zulassen.

Immediate:
Das zweite Byte der Instruktion bildet den Operanden.

Zero Page:
Das zweite Byte gibt die Adresse auf der sogenannten »Page Zero« an. Es können somit die Speicherplätze 00_{16} bis FF_{16} (256 Byte) adressiert werden.

Absolute:
Das zweite und dritte Byte geben die absolute Adresse des Operanden an, wobei das zweite Byte das niederwertigere und das dritte Byte das höherwertige Byte der Adresse darstellt.

Relative:
Es wird relativ zum Programm Counter + 2 adressiert. Der Adressierungsbereich ist -128_{10} bis $+127_{10}$ bezogen auf den Programm Counter Stand. Das Displacement steht im zweiten Byte der Anweisung.

Indirect:
Byte 2 und 3 geben die Adresse an die die Adresse des Operanden enthält.

Indirect indexed:
Bei der indirekt indizierten Adressierung wird relativ zu den Index-Registern X bzw. Y adressiert. D.h., im zweiten Byte steht ein Displacement, dessen Wert zwischen 00_{16} und FF_{16} liegt. Zusammen mit dem Inhalt eines Indexregisters wird damit die Adresse gebildet an der die Adresse des Operanden steht.

Zero Page indexed:
Hierbei bilden die Zero Page Adresse und der Inhalt eines Indexregisters die effektive Adresse des Operanden. Page Zero kann nicht verlassen werden.

Absolute indexed:
Bei der absolut indizierten Adressierung wird zur absolut angegebenen Adresse, dargestellt durch Byte 2 und 3, der Inhalt eines Indexregisters addiert. Das Ergebnis bildet die effektive Speicheradresse des Operanden.

Implied:
Der Befehl, im Maschinenkode ein Byte, enthält die Operanden selbst. Z.B. INX, inkrementiere das Indexregister X ist ein Beispiel dafür.

5.1 Architektur des Mikroprozessors 6502

Der Baustein ist an zwei internen Bussystemen orientiert, dem internen Adreßbus und dem internen Datenbus. Der interne Adreßbus bildet die Verbindung zwischen den internen Registern wie Akku, Index-Register u.a. und den Adreßleitungstreibern. Der interne Datenbus verbindet diese Register mit dem Datenbus-Zwischenspeicher und der Befehlsdekodierlogik.

Als Register stehen zur Verfügung:

Programm Counter (PC), 16 Bit breit

Akkumulator (A), 8 Bit breit. Über ihn werden alle Lade, Speicher, arithmetische und logische Operationen ausgeführt.

Index-Register (X, Y), beide 8 Bit breit. Sie enthalten praktisch das höherwertige Byte einer Adresse. Ausnahme sind die Page Zero indizierten Befehle, bei ihnen wirken sie auf das niederwertige Byte.

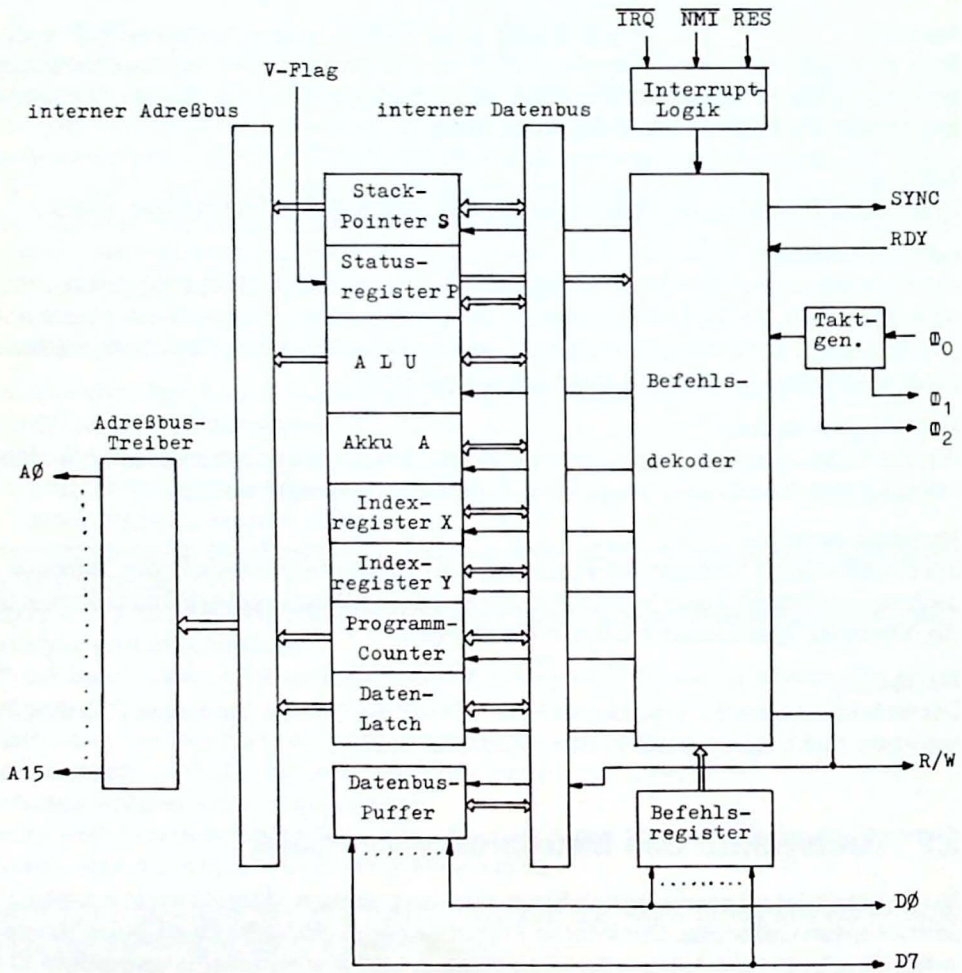


Bild 5.1.1: Innere Struktur des Mikroprozessors MCS6502

Stack Pointer (S), 8 Bit breit adressiert er den Stapelspeicher im Bereich 0100_{16} bis $01FF_{16}$.

Prozessor-Status-Register (P), 8 Bit breit enthält es die Statusflags.

N (negative): zeigt an, ob das Ergebnis einer Operation negativ oder positiv ist.

V (Overflow): gibt bei Vorzeichenarithmetik Bereichsüber- bzw. Bereichsunterschreitungen an.

Z (zero): signalisiert, das Resultat einer Verknüpfung ist Null.

C (carry): Überlaufanzeige bei arithmetischen Befehlen, Vergleichen und Schiebeinstruktionen.

D (decimal): gibt an, ob der Prozessor sich im Dezimal- oder Binär-Mode befindet.

I (interrupt): gibt Interrupt frei bzw. sperrt ihn.

B (break): wird beim BRK-Befehl auf »1« gesetzt.

Der Mikroprozessorbaustein enthält weiterhin einen Taktgenerator der aus dem externen Takt zwei weitere Taktsignale generiert und einer Interrupt-Logik die die Signale $\overline{\text{RES}}$ (Reset), $\overline{\text{IRQ}}$ (interrupt request) und $\overline{\text{NMI}}$ (non maskable interrupt) verarbeitet. Die Befehlsdekodierlogik und das Befehlsregister (es enthält seine Informationen direkt vom Datenbus) bilden die letzten beiden Elemente der Gesamtarchitektur.

In Bild 5.1.1 ist die innere Struktur in vereinfachter Form dargestellt.

5.2 Befehlsliste des 6502

Verwendete Abkürzungen bei den Adressierungsarten:

I	immediate
Z	Zero Page
A	absolute
Z, X	Zero Page indiziert
A, X	absolute indizierte
(Ind, X)	Xindizierte und indirekte
(Ind, Y)	indirekte und Y indizierte

Weitere Abkürzungen sind:

$\vee$	logisches ODER
$\wedge$	logisches UND
$\veebar$	logisches Exklusiv ODER
*	je nach Adressierungsart

Befehl	Zeit in µs* bei 1MHz	Adressierungsarten I	Z	A	Z,X	A,X	A,Y	Ind X,Y	Wirkung	Status
LDA	2 – 6	X	X	X	X	X	X	X	(M) → A	N,Z
STA	3 – 6		X	X	X	X	X	X	(A) → M	
AND	2 – 6	X	X	X	X	X	X	X	A ∧ M → A	N,Z
ORA	2 – 6	X	X	X	X	X	X	X	A ∨ M → A	N,Z
EOR	2 – 6	X	X	X	X	X	X	X	A ≠ M → A	N,Z
ADC	2 – 6	X	X	X	X	X	X	X	A + M + C → A	N,V,Z,C
SBC	2 – 6	X	X	X	X	X	X	X	A – M – C → A	N,V,Z,C
CMP	2 – 6	X	X	X	X	X	X	X	A – M → P	N,Z,C
BIT	3 und 4		X	X					A ∧ M → P	Z,M(7) → N M(6) → V
INC	2 – 4	A	X	X	X	X			A+1 → A;M+1 → M	M(7) → N,Z
DEC	2 – 4	A	X	X	X	X			A–1 → A;M–1 → M	M(7) → N,Z
ASL	2 – 4	A	X	X	X	X			C ← (7 M Ø) ← Ø	N,Z,C
LSR	2 – 4	A	X	X	X	X			Ø → (7 M Ø) → C	N = 0,Z,C
ROL	2 – 4	A	X	X	X	X			C ← (7 M Ø) ← C	N,Z,C
ROR	2 – 4	A	X	X	X	X			C → (7 M Ø) → C	N, Z, C
LDX	2 – 4	X	X	X	X		X		M → X	N,Z
LDY	2 – 4	X	X	X	X	X			M → Y	N,Z
STX	3 und 4		X	X	X				X → M	
STY	3 und 4		X	X	X				Y → M	
CPX	2 – 4	X	X	X					X – M → P	N,Z,C
CPY	2 – 4	X	X	X					Y – M → P	N,Z,C

Die nachfolgenden Befehle sind alle nur in einer Adressierungsart, der implied Adressierung, möglich.

Befehl	Zeit in µs	Wirkung	Status
INX	2	X + 1 mod. FF → X	N,Z
DEX	2	X – 1 mod. FF → X	N,Z
INY	2	Y + 1 mod. FF → Y	N,Z
DEY	2	Y – 1 mod. FF → Y	N,Z
TAX	2	A → X	N,Z
TXA	2	X → A	N,Z
TAY	2	A → Y	N,Z
TYA	2	Y → A	N,Z
TSX	2	S → X	N,Z
TXS	2	X → S	

Befehl	Zeit in µs	Wirkung	Status
PHA	3	A → M(S), S − 1 → S	
PLA	4	S + 1 → S, M(S) → A	
PHP	3	P → M(S), S − 1 → S	
PLP	4	S + 1 → S, M(S) → P	alle
CLC	2	Ø → C	C
SEC	2	1 → C	C
CLD	2	Ø → D, Binärmode	D
SED	2	1 → D, Dezimalmode	D
CLI	2	Ø → I, Interrupts erlaubt	I
SEI	2	1 → I, Interrupts sperren	I
CLV	2	Ø → V	V
NOP	2	no operation, Leerbefehl PC = PC + 1	

Bei allen bedingten Sprüngen handelt es sich um Zwei-Byte-Befehle mit relativer Adressierung.
Ausführungszeiten; kein Sprung 2 µs, Sprung 3 µs, Sprung mit Seitenwechsel 4 µs.
Der Sprung erfolgt auf PC + 2 ± Displacement

Befehl	Funktion	engl. Beschreibung
BCC	Sprung, wenn C = 0	branch if carry clear
BCS	Sprung, wenn C = 1	branch if carry set
BNE	Sprung, wenn Z = 0	branch if not equal
BEQ	Sprung, wenn Z = 1	branch if equal
BPL	Sprung, wenn N = 0	branch if plus
BMI	Sprung, wenn N = 1	branch if minus
BVC	Sprung, wenn V = 0	branch if overflow clear
BVS	Sprung, wenn V = 1	branch if overflow set

Sprünge und Rücksprung aus Interrupt bzw. Unterprogramm

Befehl	Zeit in µs	Adressierung, Länge	Funktion	Status
JMP	3	absolut, 3 Byte	Byte 2,3 → PC	
JMP	5	indirekt, 3 Byte	(Byte 2,3) → PC	
JSR	6	absolut, 3 Byte	PC → S, S − 2 → S, Byte 2,3 → PC	
BRK	7	implied, 1 Byte	P,PC →S,S − 3 → S	1 → B,I
RTS	6	implied, 1 Byte	S → PC,S + 2 → S	
RTI	6	implied, 1 Byte	S → PC,P; S + 3 → S	alle

6. Der Mikroprozessor 8085 von Intel

Der in der Industrie wohl meist verwendete 8 Bit Mikroprozessor ist der 8085 (8080) von Intel. Man kann den 8085 als verbesserten 8080A auffassen. Er ist voll softwarekompatibel.

Sein großes Plus gegenüber dem 8080 ist, daß er nur mit einer Versorgungsspannung auskommt und die interne Erzeugung sämtlicher Steuersignale. Doch der Vergleiche genug, beschäftigen wir uns mit dem realen 8085.

Daten- und Adreßbus arbeiten Zeitmultiplex. Die niederwertigeren acht Adreßbits werden über den Datenbus übertragen. Die 16 Bit Adresse muß also zwischengespeichert werden. Dafür existiert das Signal ALE (address latch enable). Es zeigt an, daß die Adresse auf Adreß- und Datenbus verfügbar ist.

Der 8085 bietet ausgezeichnete Interruptmöglichkeiten durch seine fünf Interrupteingänge. Drei davon, RST5.5, RST6.5 und RST7.5 veranlassen die Ausführung der RST- (Restart) Instruktion mit fest zugewiesenen Startadressen. Für diese Eingänge besteht eine feste Prioritätsstruktur, außerdem sind sie einzeln maskierbar. Ein weiterer Interrupteingang ist das Signal »TRAP«, was soviel wie »Falle« bedeutet. Diese Unterbrechung kann nicht maskiert werden und wird meist bei der Fehlersuche zusammen mit externer Logik als Hardwarebreakpoint benutzt. Der entsprechende Interruptvektor zeigt in diesem Fall z. B. auf den »DEBUG«. Unter DEBUG versteht man ein Hilfsprogramm, das die Fehlersuche unterstützt.

Wie beim SCMP steht auch beim 8085 ein serieller Eingabe- und Ausgabe-Port zur Verfügung.

Der Mikroprozessor 8085 ist DMA-fähig, d. h. er kann bei DMA z. B. über den »HOLD«-Eingang gestoppt werden. Adreß- und Datenbus nehmen dann den »Tristate« (hochohmigen) Zustand an und das externe Gerät kann mit dem Speicher verkehren.

Zusammenfassung der wichtigsten Daten des 8085:

- 8 Bit Datenbus gemultiplext mit den niederwertigen 8 Bit der Adresse.
- 16 Bit Adreßbus, wobei die unteren 8 Bit mit dem Datenbus gemultiplext werden.
- 7 sogenannte »general purpose« Register (Register für allgemeine Verwendung).
- eine einzige Spannungsversorgung von 5V.
- Taktoszillator auf dem Baustein integriert.
- Die maximal mögliche Taktfrequenz beträgt 3,13 MHz.
- Befehlsausführungszeit zwischen 1,2 und 5,8 Mikrosekunden.
- Es stehen je ein serieller Eingabe- und Ausgabeport zur Verfügung.
- 5 Interrupt-Eingänge mit Prioritätsstruktur und Maskierungsmöglichkeiten.
- Da der Stack extern im RAM liegt, ist die Unterprogrammverschachtelung nur durch die RAM-Größe beschränkt.
- Dem Anwender stehen 80 Grundbefehle zur Verfügung.

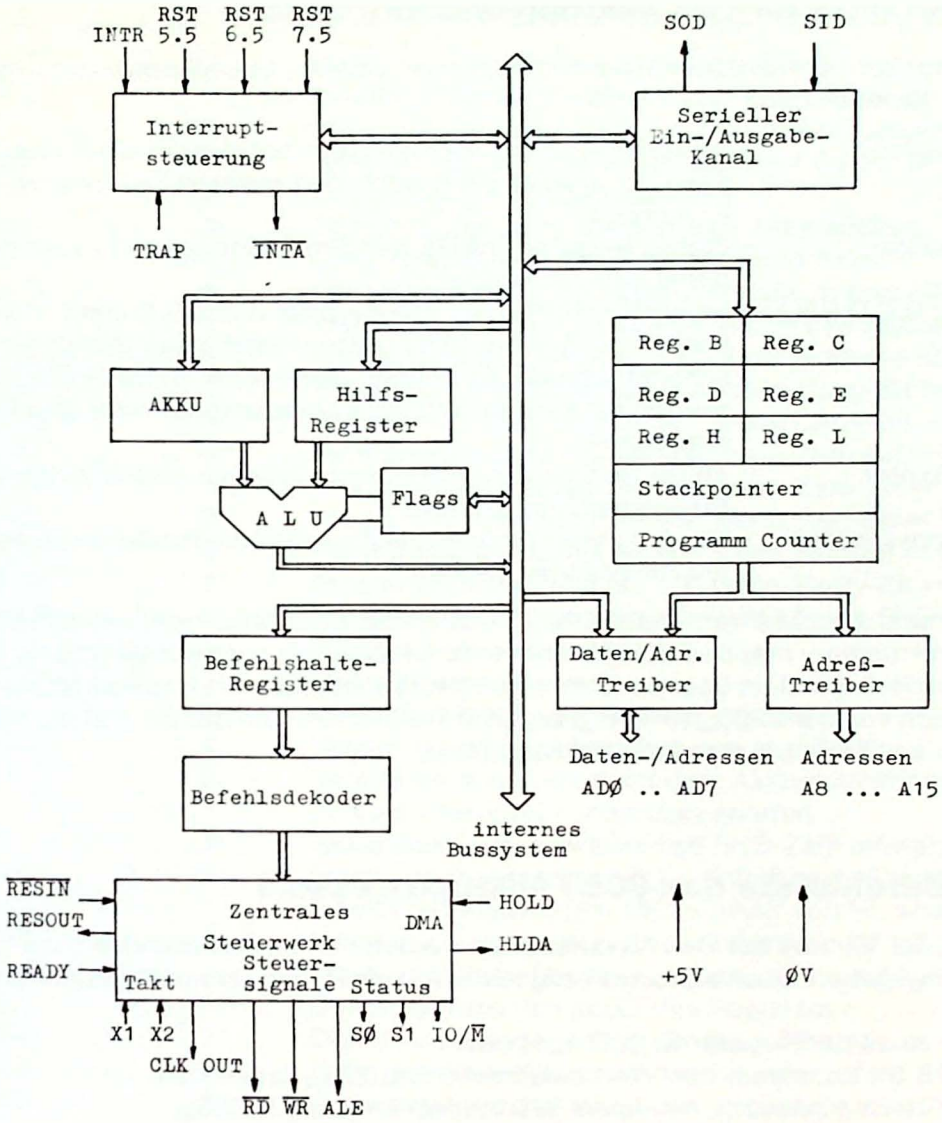


Bild 6.1.1: Innere Struktur des Mikroprozessors 8085

6.1 Architektur des Mikroprozessors 8085

Insgesamt hat der 8085 zehn Register, von denen sieben generell verwendet werden können. Es sind dies:

Akkumulator	– zentrales Register, alle Daten Ein- und Ausgabe läuft über den Akku. Er enthält auch nach den meisten Operationen das Ergebnis.
Flag-Register	– dieses Register enthält den Prozessorstatus. Es kann nicht direkt geladen werden.
Register B,C,D,E,H,L	– jedes ist acht Bit breit. Die Register B und C, D und E, H und L können zu je einem 16 Bit breiten Registerpaar zusammengeschaltet werden. Es können somit auch 16-Bit-Operationen ausgeführt werden. Indizierte Adressierung immer über Registerpaar H,L.
Stack-Pointer	– 16 Bit Register das die oberste Adresse »top of stack« des Stapelspeichers enthält.
Programm-Counter	– Der Programm Counter zeigt auf die Adresse des als nächsten auszuführenden Befehls.

Die Ein- und Ausgabe von und zur Peripherie erfolgt beim 8085 nicht mit dem sogenannten »memory mapped I/O« welcher externe Elemente wie normale Speicherzellen behandelt, sondern über bestimmte Ein-/Ausgabekanäle. Die Kanalnummer kann im Bereich von 0 bis 255_{10} ($0 - FF_{16}$) liegen. Im nachfolgenden Bild 6.1.1 ist die innere Struktur des Mikroprozessors 8085 dargestellt.

6.2 Befehlsliste des 8085 Mikroprozessors

Wegen der Vielzahl der Befehlsvariationen werden nur die Basisbefehle ohne Hex-Kode angegeben. Doch zunächst einige Erklärungen zu den verwendeten Abkürzungen.

r = eines der Register B, C, D, E, H oder L
d = 8 Bit Konstante den Wert zwischen 0 und 255_{10} liegen kann
dd = 16 Bit Konstante mit einem Wert zwischen 0 und 65535_{10}
k = Konstante k ($0 \leq k \leq 7$) ist im Befehlskode enthalten
nr = Ein-/Ausgabekanal-Nr. Wert zwischen 0 und 255_{10}
adr = steht für eine 16 Bit Adresse
m = Memory, es wird die über das H,L-Registerpaar adressierte Speicherstelle angesprochen
SP = Stack-Pointer
PSW = Programm-Status-Wort (Flag-Register)
Bei allen 16-Bit-Werten (Adressen, Daten) steht immer das niederwertige Byte zuerst.

Befehl	Bytes	Beschreibung des Befehls
ADD r	1	Inhalt von Register r zum Akku addieren
ADD m	1	Inhalt der Speicherstelle m zum Akku addieren
ADC r	1	Inhalt von Register r und Carry zum Akku addieren
ADC m	1	gleiche Funktion jedoch bezogen auf Speicher m
ADI d	2	Datum d wird zum Akku addiert
ACI d	2	Datum d und Carry werden zum Akku addiert
ANA r	1	UND-Verknüpfung von Register r und Akku
ANA m	1	UND-Verknüpfung von Speicherstelle m und Akku
ANI d	2	UND-Verknüpfung des Wertes d und des Akkus
CALL adr	3	Fortsetzung des Programms bei der Adresse adr
CC adr	3	Programmfortsetzung bei adr, wenn Carry-Bit = 1
CNC adr	3	Programmfortsetzung bei adr, wenn Carry-Bit = 0
CZ adr	3	Programmfortsetzung bei adr, wenn Zero-Bit = 1
CNZ adr	3	Programmfortsetzung bei adr, wenn Zero-Bit = 0
CM adr	3	Programmfortsetzung bei adr, wenn Sign-Bit = 1
CP adr	3	Programmfortsetzung bei adr, wenn Sign-Bit = 0
CPE adr	3	Programmfortsetzung bei adr, wenn Parity-Bit = 1
CPO adr	3	Programmfortsetzung bei adr, wenn Parity-Bit = 0
CMA	1	Negieren des Akkuinhalts
CMC	1	Carry-Bit komplementieren
CMP r	1	Vergleiche Inhalt von Register r mit Akku
CMP m	1	Vergleich von Speicherinhalt von m mit Akku
CPI d	2	Vergleiche den Wert d mit dem Akkumulatorinhalt
DI	1	Disable Interrupts – Interrupts sperren
DAA	1	Akkuinhalt in eine zweistellige BCD-Zahl umwandeln
DAD B	1	Inhalt von Registerpaar BC zu Inhalt von HL addieren
DAD D	1	Inhalt von Registerpaar DE zu Inhalt von HL addieren
DAD H	1	Inhalt von Registerpaar HL zu Inhalt von HL addieren
DAD SP	1	Inhalt von Stack-Pointer zu Inhalt von HL addieren
DCR r	1	Dekrementiere den Inhalt des Registers r
DCR m	1	Dekrementiere den Inhalt der Speicherstelle m
DCX B	1	Dekrementiere den Inhalt des Registerpaares BC
DCX D	1	Dekrementiere den Inhalt des Registerpaares DE
DCX H	1	Dekrementiere den Inhalt des Registerpaares HL
DCX SP	1	Dekrementiere den Inhalt des Stack-Pointers
EI	1	Enable Interrupts – Interrupt erlauben
HLT	1	HALT, das Programm hält an bis ein Interrupt auftritt
IN nr	2	Akku mit dem Inhalt des Eingabekanals nr laden
INX B	1	Inkrementiere den Inhalt des Registerpaares BC
INX D	1	Inkrementiere den Inhalt des Registerpaares DE
INX H	1	Inkrementiere den Inhalt des Registerpaares HL

Befehl	Bytes	Beschreibung des Befehls
INX SP	1	Inkrementiere den Inhalt des Stack-Pointers
INR r	1	Inkrementiere den Inhalt des Registers r
INR m	1	Inkrementiere den Inhalt der Speicherstelle m
JMP adr	3	Sprung zur Adresse adr
JC adr	3	Sprung zur Adresse adr, wenn Carry-Bit = 1
JNC adr	3	Sprung zur Adresse, wenn Carry-Bit = 0
JZ adr	3	Sprung zur Adresse adr, wenn Zero-Bit = 1
JNZ adr	3	Sprung zur Adresse adr, wenn Zero-Bit = 0
JP adr	3	Sprung zur Adresse adr, wenn Sign-Bit = 0
JM adr	3	Sprung zur Adresse adr, wenn Sign-Bit = 1
JPE adr	3	Sprung zur Adresse adr, wenn Parity-Bit = 1
JPO adr	3	Sprung zur Adresse adr, wenn Parity-Bit = 0
LDA adr	3	Lade Akku mit dem Inhalt der Adresse adr
LDAX B	1	Akku mit Inhalt des durch BC adressierten Bytes laden
LDAX D	1	Akku mit Inhalt des durch DE adressierten Bytes laden
LHLD adr	3	Registerpaar HL mit Inhalt der Adresse adr und adr + 1 laden
LXI B, dd	3	Lade Registerpaar BC mit Wert dd
LXI D, dd	3	Lade Registerpaar DE mit Wert dd
LXI H, dd	3	Lade Registerpaar HL mit Wert dd
LXI SP, dd	3	Lade Stack-Pointer mit dem Wert dd
MOV r1, r2	1	Lade Register r1 mit Inhalt von Register r2
MOV m, r	1	Lade Speicherstelle m mit Inhalt von Register r
MOV r, m	1	Lade Register r mit Inhalt der Speicherstelle m
MVI r, d	2	Lade Register r mit dem Wert d
MVI m, d	2	Lade Speicherstelle m mit dem Wert d
NOP	1	No operation – Leerbefehl der den PC um eins erhöht
OUT nr	2	Akkuinhalt auf Ausgabekanal mit der Nummer nr ausgeben
ORA r	1	ODER-Verknüpfung von Register r und Akkuinhalt
ORA m	1	ODER-Verknüpfung von Speicherstelle m und Akkuinhalt
ORI d	2	Akkuinhalt mit dem Wert d ODER verknüpfen
PCHL	1	Inhalt von Registerpaar HL in PC transferieren
PUSH B	1	Registerpaar BC in den Stack schreiben
PUSH D	1	Registerpaar DE in den Stack schreiben
PUSH H	1	Registerpaar HL in den Stack schreiben
PUSH PSW	1	Akkuinhalt und Programm-Status-Wort ist in den Stack bringen

Befehl	Bytes	Beschreibung des Befehls
POP B	1	Registerpaar BC von Stack laden
POP D	1	Registerpaar DE von Stack laden
POP H	1	Registerpaar HL von Stack laden
POP PSW	1	Akku und Programm-Status-Wort von Stack laden
RIM	1	Interruptmaske in den Akku lesen
RST k	1	$0 \leq k \leq 7$ Programm wird auf der Adresse $8 \cdot k$ fortgesetzt
RET	1	Rücksprung in das aufrufende Programm, Adr. im Stack
RC	1	Rücksprung, wenn das Carry-Bit = 1
RNC	1	Rücksprung, wenn das Carry-Bit = 0
RZ	1	Rücksprung, wenn das Zero-Bit = 1
RNZ	1	Rücksprung, wenn das Zero-Bit = 0
RM	1	Rücksprung, wenn das Sign-Bit = 1
RP	1	Rücksprung, wenn das Sign-Bit = 0
RPE	1	Rücksprung, wenn das Parity-Bit = 1
RPO	1	Rücksprung, wenn das Parity-Bit = 0
RLC	1	Akku um 1 Bit nach links schieben, $C \leftarrow$ Bit $2^7 \leftarrow$ Bit $2^0 \leftarrow \emptyset$
RRC	1	Akku um 1 Bit nach rechts schieben, $\emptyset \rightarrow$ Bit $2^7 \rightarrow$ Bit $2^0 \rightarrow C$
RAL	1	Akku um eins nach links rotieren, $C \leftarrow$ Bit $2^7 \leftarrow$ Bit $2^0 \leftarrow C$
RAR	1	Akku um eins nach rechts rotieren, $C \rightarrow$ Bit $2^7 \rightarrow$ Bit $2^0 \rightarrow C$
SIM d	2	Interruptmaske auf den Wert d setzen
STA adr	3	Akkuinhalt in die Adresse adr schreiben
STAX B	1	Akkuinhalt in das durch BC adressierte Byte schreiben
STAX D	1	Akkuinhalt in das durch DE adressierte Byte schreiben
SHLD adr	3	Registerpaar HL unter Adresse adr und adr + 1 abspeichern
SPHL	1	Stack-Pointer mit Inhalt des Registerpaares HL laden
SUB r	1	Inhalt des Registers r vom Akkuinhalt subtrahieren
SUB m	1	Inhalt des Speichers m vom Akkuinhalt subtrahieren
SBB r	1	Inhalt von r und Carry-Bit vom Akku subtrahieren
SBB m	1	Inhalt von m und Carry-Bit vom Akku subtrahieren
SUI d	2	Wert d vom Akkumulatorinhalt subtrahieren
SBI d	2	Wert d und Wert des Carry-Bit vom Akku subtrahieren
STC	1	Carry-Bit setzen
XCHG	1	Vertauscht Inhalt der Registerpaare DE und HL

Befehl	Bytes	Beschreibung des Befehls
XTHL	1	Vertauscht Inhalt von HL und Inhalt des durch Stack adressierten Wortes
XRA r	1	Akkuinhalt mit Inhalt von Register r Exklusiv-ODER-verknüpfen
XRA m	1	Speicherinhalt von m mit Akkuinhalt Exklusiv-ODER-verknüpfen
XRI d	2	Akkuinhalt mit dem Wert d Exklusiv-ODER-verknüpfen

7. Der Mikroprozessor MC6809 von Motorola

Einer der leistungsfähigsten 8 Bit Mikroprozessoren ist der MC6809 von Motorola. Es ist ein Baustein der dritten Generation der Motorola 6800 Familie. Für verschiedene Einsatzgebiete stehen Ausführungen mit unterschiedlichen Taktfrequenzen zur Verfügung. 6809 mit 1 MHz. 68A09 mit 1,5 MHz und 68B09 mit 2 MHz.
Alle Komponenten der Bausteinfamilie 6800 können angeschlossen werden. Die Software ist aufwärtskompatibel, d.h. die Programme des 6800 »laufen« auch auf dem 6809.
Der Prozessor kann in einem 40-poligen Kunststoff oder Keramikgehäuse geliefert werden.
Die zehn möglichen Adressierungsarten sind:

Inherent – Operationskode enthält die Adreßinformation, schließt die Akkumulatoren ein.

Immediate – Die dem OP-Kode folgenden 8 oder 16 Bit bilden den Operanden. Wird auch als unmittelbare Adressierung bezeichnet.

Extended – Die dem Operationskode folgenden beiden Byte stellen die absolute 16 Bit Adresse des Operanden dar.

Extended indirect – Die folgenden 16 Bit stellen die Adresse dar an der die absolute Adresse des Operanden steht.

Direct – Die dem Befehl folgenden 8 Bit bilden das niederwertige Byte der Operandenadresse, das höherwertige Byte wird durch den Inhalt des sogenannten »direct page register« geliefert.

Register – Hierbei werden Register oder die Akkumulatoren angesprochen und kein Speicher.

Indexed – Es wird über die Indexregister X, Y, U und S adressiert.

Indexed indirect – Wie bei der indizierten Adressierung jedoch steht an der durch die Register spezifizierten Speicherstelle wiederum eine Adresse; die effektive Adresse des Operanden.

Relativ – Diese Adressierungsart wird bei Sprüngen und Verzweigungen angewendet. Die dem Befehl folgenden Byte (1 oder 2) bilden ein Displacement in Zweierkomplementdarstellung.

PC-relativ – Programmzähler relativ bedeutet, daß mittels 8 oder 16 Bit Offset relativ zum PC adressiert wird. Kann auch als indirekt ausgeführt werden.

Zusammenfassung der wichtigsten Daten des MC6809:

Bausteinfamilie des 6800 verwendbar
Softwarekompatibel zum 6800
2 Index-Register je 16 Bit, 2 Stack-Pointer je 16 Bit
2 Akkumulatoren je 8 Bit die zu einem 16 Bit Akkumulator zusammengeschaltet werden können.
»direct page register« erlaubt direkte (schnelle) Adressierung im gesamten Speicherbereich
Taktgenerator auf dem Baustein integriert
DMA und Speicher-»refresh« Betrieb möglich
3 Interruptarten, nicht maskierbarer Interrupt, Interrupt mit Retten des gesamten Prozessorstatus und schneller Interrupt bei dem nur der Programm Counter und das Bedingungskoderegister gerettet werden.
Möglichkeit mit langsamen Speichern oder Peripherie zu arbeiten
Prozessor gibt Interrupt-Quittungssignal aus, damit ist eine Vektorisierung durch externe Einheiten möglich
externe Synchronisation ist möglich
einfache 5 V Spannungsversorgung
nicht maskierbare Interrupt werden nach »RESET« bis zum ersten Laden des Stackpointers S blockiert.
10 Adressierungsarten
unterstützte Stack-Manipulation
8×8 Bit vorzeichenlose Multiplikation, 16 Bit Arithmetik
die 59 Basisbefehle lassen 1464 Variationen von unterschiedlichen Instruktionen mit verschiedenen Adressierungsarten zu
Befehlsausführungszeiten der 2 MHz-Version zwischen 1 und 10 µs (im Mittel 3 µs)
alle Daten- und Adreßleitungen sind herausgeführt

7.1 Architektur des Mikroprozessors 6809

Der Baustein ist in NMOS-Technologie hergestellt und ist im 40poligen Plastik- und Keramikgehäuse verfügbar.
Die beiden 8 Bit Akkumulatoren A und B lassen sich bei einigen Instruktionen zu einem 16 Bit Akkumulator D zusammenschalten, wobei der Akku A das höherwertige Byte darstellt.

Mit Hilfe des »Direct Page Register« (DP), welches das höherwertige Adreßbyte bildet, kann im gesamten Adreßraum die direkte Adressierung angewendet werden. Das bringt den Vorteil einer kürzeren Befehlsausführungszeit.
Die 16 Bit Index-Register X und Y bilden die Basisadresse bei der indizierten Adressierung.

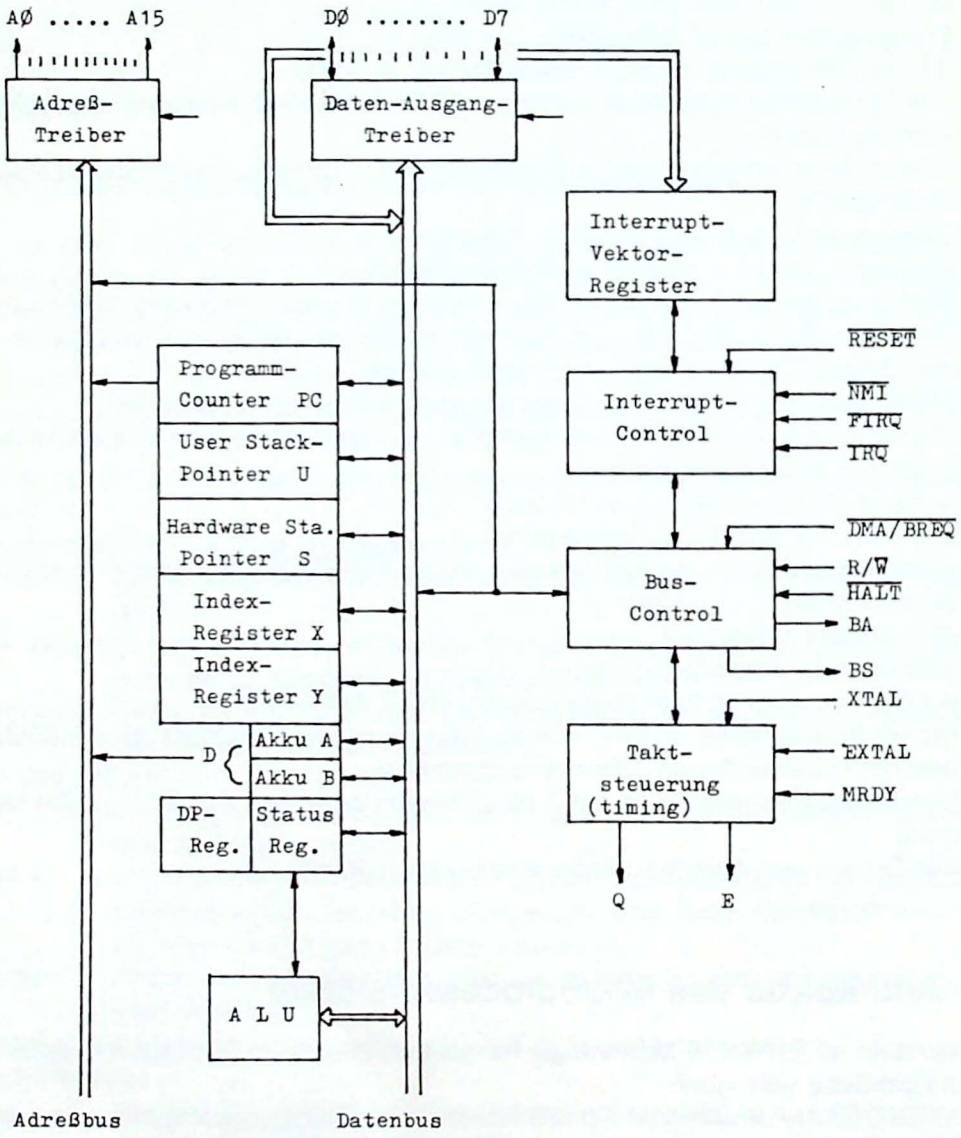


Bild 7.1.1: Architektur des Mikroprozessors MC6809

Mit Hilfe der Stackpointer S (Hardware-Stack-Pointer) und U (User-Stack-Pointer) können sehr komfortable Stacktechniken realisiert werden. Stack-Pointer S wird z. B. beim Unterprogrammaufruf automatisch vom Prozessor benutzt um den Prozessorstatus zu retten, während der Stack-Pointer U allein dem Programmierer zur Verfügung steht. Das Bedingungskoderegister enthält Bits (Flags) für Carry, Overflow, Zero, Negative, IRQ-Mask , Half Carry, FIRQ-Mask und Entire-Flag.

7.2 Befehlsliste des 6809 in Instruktionstypen aufgeteilt

16 Bit Akku- und Speicheroperationen

Mnemonic	Operation
ADDD	Addiert den Inhalt eines Speicherwortes zum D-Akku
CMPD	Vergleicht Inhalt eines Speicherwortes mit dem D-Akku
EXG D, r	Austausch von Register r (X, Y, U, S, PC) und D-Akku
LDD	Akku D mit dem Inhalt eines Speicherwortes laden
SEX	Akku B auf Akku A erweitern
STD	D-Akku in den Speicher schreiben
SUBD	Inhalt eines Speicherwortes vom Akku D subtrahieren
TFR D, r	Akku nach Register r (X, Y, U, S, PC) schreiben
TFR r, D	Register r nach Akku D transferieren

8 Bit Akku- und Speicheroperationen

Mnemonic	Operation
ADCA, ADCB	Speicherinhalt und Carry zum Akku A oder B addieren
ADDA, ADDB	Speicherinhalt zum Akkumulator A oder B addieren
ANDA, ANDB	UND-Verknüpfung von Speicherinhalt mit Akku A oder B
ASL, ASLA, ASLB	arithmetisch links schieben von Speicher, Akku A oder B
ASR, ASRA, ASRB	arithmetisch rechts schieben von Speicher, Akku A oder B
BITA, BITB	Bit im Speicher mit Bit im Akku A oder B vergleichen
CLR, CLRA, CLRB	Speicherinhalt, Akku A oder Akku B löschen
CMPA, CMPB	Vergleich von Speicherinhalt und Akku A oder B
COM, COMA, COMB	Speicherinhalt, Inhalt von Akku A oder B komplementieren
DAA	Dezimalkorrektur auf den Inhalt von Akkumulator A anwenden
DEC, DECA, DECB	Dekrementieren von Speicherinhalt, Inhalt Akku A oder B
EORA, EORB	Speicherinhalt mit Akku A oder B Exklusiv-ODER-verknüpfen
EXG r1, r2	Austausch von Register r1 mit r2 (r1, r2 = A, B, CC oder DP)
INC, INCA, INCB	Inkrementieren von Speicherinhalt, Inhalt Akku A oder B
LDA, LDB	Akku A oder B mit Speicherinhalt laden
LSL, LSLA, LSLB	logisch links schieben von Speicher, Akku A oder B

LSR, LSRA, LSRB	logisch rechts schieben von Speicher, Akku A oder B
MUL	Vorzeichenlose Multiplikation von Akku A und B, Ergebnis steht in Akku D
NEG, NEGA, NEGB	Negieren des Speicherinhaltes, Inhalt Akku A oder B
ORA, ORB	Speicherinhalt mit Akku A oder B ODER-verknüpfen
ROL, ROLA, ROLB	Speicherinhalt oder Inhalt Akku A oder B links rotieren
ROR, RORA, RORB	Speicherinhalt oder Akku A oder B rechts rotieren
SBCA, SBCB	Speicherinhalt und Carry von Akku A oder B subtrahieren
STA, STB	Inhalt von Akku A oder B in Speicher schreiben
SUBA, SUBB	Speicherinhalt von Akkuinhalt (A oder B) subtrahieren
TST, TSTA, TSTB	Testen einer Speicherzelle oder eines Akkus A oder B
TFR r1, r2	Inhalt von Register r1 nach Register r2 transferieren (r1, r2 = A, B, CC, DP)

Index-Register und Stack-Pointer Befehle

Mnemonic	Operation
CMPS, CMPU	Speicher mit Stackpointer S oder U vergleichen
CMPX, CMPY	Speicher mit Indexregister X oder Y vergleichen
EXG r1, r2	Austausch von Register r1 mit r2 (r1, r2 = D, X, Y, S, U, PC)
LEAS, LEAU	Effektive Adresse in den Stackpointer S oder U bringen
LEAX, LEAY	Effektive Adresse in Indexregister X oder Y schreiben
LDS, LDU	Stackpointer S oder U mit Speicherinhalt laden
LDX, LDY	Indexregister X oder Y mit Speicherinhalt laden
PSHS, PSHU	Transferiert ein oder alle Register in den Stack (S, U)
PULS, PULU	Holt ein oder alle Register aus dem Stack S oder U
STS, STU	Stackpointer S oder U in den Speicher schreiben
STX, STY	Indexregister X oder Y in den Speicher schreiben
TFR r1, r2	Transferiert Register r1 zu r2 (r1, r2 = D, X, Y, S, U, PC)
ABX	Vorzeichenlose Addition des Akku B zum Indexregister X

Verschiedene Befehle

Mnemonic	Operation
ANDCC d	UND-Verknüpfung des Wertes d mit Bedingungsregister CC
CWAI d	UND-Verknüpfung des Wertes d mit dem Register CC und danach auf Interrupt warten
NOP	Leerbefehl (no operation) erhöht den PC um eins
JMP	Unbedingter Sprung zur effektiven Adresse
JSR	Sprung in Unterprogramm, bewirkt Retten des Prozessorstatus im Stack S
RTI	Rücksprung aus Interrupt-Routine, Prozessorstatus aus Stack S holen

RTS	Rücksprung aus Unterprogramm, sonst wie RTI
SWI, SWI2, SWI3	Softwareinterrupt, Sprung in UP (Adr. in FFFF → FFFØ)
SYNC	Prozessor wird mit IR-Leitung extern synchronisiert

Verzweigungsbefehle

Alle Verzweigungsbefehle (bedingte Sprünge) benutzen einen 8 oder 16 Bit (L = long) Offset in Zweierkomplementdarstellung. Dem Verzweigungsbefehl muß eine Vergleichsoperation oder eine Operation die das Statusregister CC verändert vorangegangen sein. Es wird verzweigt, wenn die Bedingung eintrifft (also »wahr« ist). Aus diesem Grund sind in der Spalte »Operation« die Bedingungen in komprimierter Form dargestellt. Zum besseren Verständnis der Mnemonic steht zusätzlich die Operation in englisch dabei.

Mnemonic	Operation	
BCC, LBCC	branch if carry clear	(carry = 0)
BCS, LBCS	branch if carry set	(carry = 1)
BEQ, LBEQ	branch if equal	(OP1 = OP2)
BGE, LBGE	branch if greater than or equal	(OP1 ≧ OP2) mit Vorzeichen
BGT, LBGT	branch if greater	(OP1 > OP2) mit Vorzeichen
BHI, LBHI	branch if higher	(OP1 > OP2) ohne Vorzeichen
BHS, LBHS	branch if higher or same	(OP1 ≧ OP2) ohne Vorzeichen
BLE, LBLE	branch if less than or equal	(OP1 ≦ OP2) mit Vorzeichen
BLO, LBLO	branch if lower	(OP1 < OP2) ohne Vorzeichen
BLS, LBLS	branch if lower or same	(OP1 ≦ OP2) ohne Vorzeichen
BLT, LBLT	branch if less than	(OP1 < OP2) mit Vorzeichen
BMI, LBMI	branch if minus	(sign = 1)
BNE, LBNE	branch if not equal	(OP1 ≠ OP2)
BPL, LBPL	branch if plus	(sign = 0)
BRA, LBRA	branch always	(immer verzweigen)
BRN, LBRN	branch never	(niemals verzweigen)
BSR, LBSR	branch to subroutine	(in Unterprogramm verzweigen)
BVC, LBVC	branch if overflow clear	(overflow = 0)
BVS, LBVS	branch if overflow set	(overflow = 1)

8. Der Mikroprozessor Z80 von Zilog

Dieser Mikroprozessor entspricht in seiner Struktur weitgehend dem 8080 bzw. 8085 und ist mit diesem fast vollständig Softwarekompatibel. Der Z80 kennt 158 Befehle die die Instruktionen des 8085 einschließen. Vom Befehlsatzumfang ist er also fast doppelt so leistungsfähig wie der 8085. Die Register A, B, C, D, H und L sind doppelt vor-

handen. Zusätzlich besitzt er noch ein Speicher-Auffrisch-Register über das sich dynamische RAM's vom Prozessor auffrischen lassen ohne das die Befehlsausführung beeinträchtigt wird.
Die Befehlsausführungszeit liegt bei einer Taktfrequenz von 4 MHz (4 MHz-Version des Bausteins) zwischen 1 und 5,75 Mikrosekunden.

Zusammenfassung der wichtigsten Daten des Z80:

8 Bit Datenbus
16 Bit Adreßbus
17 interne Register
einzelne 5V Spannungsversorgung
drei Interruptarten mit einem zusätzlichen nicht maskierbaren Interrupt
158 Befehle die alle 78 Instruktionen des 8080 einschließen, ausgenommen sind die Befehle SIM und RIM des 8085
leistungsfähige 16-Bit-Befehle
Maximale Taktfrequenz von 4 MHz
Befehlsausführungszeiten zwischen 1 und 5,75 Mikrosekunden bei der 4 MHz-Version
Lieferung im 40poligen Gehäuse

9. Vor- und Nachteile der besprochenen Mikroprozessoren

Die geschilderten Vor- und Nachteile spiegeln die Meinung des Verfassers wieder und beruhen auf dessen Erfahrung mit den genannten Prozessoren.
Befassen wir uns zuerst mit dem INS8060 (SCMP) von National Semiconductor.
Dieser Mikroprozessor eignet sich auf Grund seiner einfachen Struktur, seines geringen Preises und des Befehlsvorrats besonders für einfache Anwendungen wie z.B. eine Heizungssteuerung. Auch als Steuerprozessor für z.B. einen Drucker oder Terminals ist er recht gut geeignet.
Wofür er in der Praxis nicht geeignet ist, das sind Anwendungen bei denen es auf hohe Geschwindigkeiten (Realtimeverarbeitung) ankommt und wenn sehr viele Rechenoperationen auszuführen sind.
Leider gibt es recht wenig Softwareunterstützung für diesen Mikroprozessor.
Wesentlich leistungsfähiger ist der 6502 der in den Mikrocomputern KIM, SIM, AIM und CBM verwendet wird. Der Baustein ist von der Firma Rockwell. Er kann praktisch auf allen Gebieten, die eine Mikrocomputerunterstützung verlangen, eingesetzt werden.
Der Nachteil des 6502, die geringe Anzahl von Registern insbesondere die nur 8 Bit breiten Indexregister, wird durch seinen wirkungsvollen und klaren Befehlssatz und seine vielfältigen Adressierungsarten wieder ausgeglichen. Software wird in großer Menge auf dem Markt angeboten.
Der Hauptvorteil des 8085A von Intel (bzw. SAB8085 von Siemens) liegt weniger im Prozessor als in der Unterstützung durch Intel. Es stehen große Mengen von Software zur Verfügung. Ebenso bietet Intel ausgereifte Entwicklungssystemkomponenten zur

Programmentwicklung an. Außerdem gibt es für die 8080/8085 Bausteinfamilie die vielfältigsten Interfacebausteine. Besonders hervorzuheben ist die leistungsstarke Interruptstruktur des 8085A.
Als Nachteil kann angeführt werden, daß es solche unterschiedlichen Adressierungsarten wie bei den anderen Prozessoren praktisch nicht gibt und der von der Mnemonic etwas schwierige bzw. gewöhnungsbedürftige Befehlssatz. Als Indexregister kann nur das H, L-Registerpaar verwendet werden.
Einsatzgebiet des 8085 wie beim 6502.
Den MC6809 von Motorola kann man eigentlich nicht direkt mit den oben genannten Mikroprozessoren vergleichen. Er ist ungleich leistungsfähiger. In erster Linie bedingt durch die jüngere Generation kann diese Steigerung erklärt werden, denn alle neuen Prozessoren sind in der Regel leistungsfähiger als die vorangegangen. Dieser Trend hält an. Doch gleich ein Wermutstropfen. Die Unterstützung durch Soft- und Hardware ist nicht so gut wie beim 8085. Auf Grund seiner Struktur und seines Befehlsvorrates ist der 6809 besonders für komplexe Prozeßsteuerungen, Micro-/Mini-Computer und »intelligente« Datenendgeräte geeignet. Seine Stackstruktur bietet nicht zu übersehende Vorteile für den Einsatz von höheren Programmiersprachen (PL/M, PASCAL, FORTRAN).

Ähnlich wie der 8085A ist auch der Z80 von Zilog einzuordnen. Mit dem erweiterten Register- und Befehlssatz, dem vektorisierbaren Interrupt und der »on chip refresh logic« für dynamische RAM's ist er jedoch von der Leistung her besser.

Ausführliche Daten und Applikationsvorschläge liefern die entsprechenden Distributoren der einzelnen Hersteller.

10. Kurzvorstellung einiger 16-Bit Mikroprozessoren

In diesem Kapitel sollen kurz die 16-Bit Mikroprozessoren 8086 von Intel, Z8000 von Zilog, MC68000 von Motorola und TMS9900 von Texas Instruments vorgestellt werden.

10.1 Der 8086 von Intel

Dieser in HMOS-Technologie aufgebaute Baustein war einer der ersten 16-Bit Mikroprozessoren die auf dem Markt kamen. Er ist dementsprechend auch nicht ganz so leistungsfähig wie z. B. der 68000 oder der Z8000. Als Zweitlieferant steht SIEMENS zur Verfügung. Der 8086 ist von der Struktur dem 8085 sehr ähnlich, denn er ist so konzipiert, daß eine Erweiterung eines 8085-Systems auf den 8086 relativ einfach zu verwirklichen ist. Die Software ist zum größten Teil aufwärtskompatibel.

Kurzdaten des 8086:
1 MByte Adreßraum
5/8 MHz Taktfrequenz
40poliges DIL-Gehäuse
Co-Prozessoren (z.B. Floatingpointprozessor 8087)
zwei Betriebsmodi – Minimalmode mit 64 KByte Adreßraum
– Maximalmode mit 1 MByte Adreßraum
4 16-Bit-Register die auch als 8 einzelne 8-Bit-Register benutzt werden können
zwei 16-Bit Pointerregister
zwei Indexregister
16-Bit-Statusregister welches die Flags des 8080 enthält
vier 16-Bit Segmentregister die auf das Kode-Segment,
Daten-Segment,
Stack-Segment,
Extra-Segment zeigen
Daten und Instruktionen müssen nicht wortorientiert im Speicher abgelegt sein
6 Adressierungsarten
Instruktionslänge zwischen 1 und 6 Byte
Datentypen: BCD, Byte, Worte (2 Byte), Pointer (4 Byte), Byteketten und Wortketten (Strings)

Leistungsstarke Entwicklungshilfen stehen zur Verfügung.

10.2 Der Z8000 von Zilog

Dieser Baustein ist im Gegensatz zum 8086 nicht mikroprogrammiert, sondern »fest verdrahtet«. Die Entwicklungshilfen sind nicht so umfangreich vorhanden wie beim 8086.
Der Z8000 soll nicht als Erweiterung des Z80 gelten. Er hat aus dem Grund eine neue Struktur.
Zweitlieferanten sind AMD, SHARP und SGS.
Es gibt zwei Versionen des Z8000. Den Z8002 (nicht segmentiert) mit 64 KByte (erweiterbar auf 384 KByte) Adreßraum und den Z8001 (segmentiert) mit 8 MByte (erweiterbar auf 48 MByte) Adreßraum.
Kurzdaten des Z8001:
NMOS-Technologie
8 MByte Adreßraum
4/6 MHz Taktfrequenz
einfache 5V Spannungsversorgung
48poliges DIL-Gehäuse
zwei Operationsmodi »privileged« und »non-privileged« (im zweiten Mode sind nicht alle Interrupts erlaubt)
Refresh-Logik für dynamische RAM's auf dem Chip integriert

Slave-Prozessoren anschließbar
16 Vielzweckregister je 16 Bit breit, davon können die ersten acht Register nochmals in je zwei 8-Bit-Register geteilt werden. Außerdem können die 16 Register zu acht 32-Bit- oder vier 64-Bit-Register zusammengeschaltet werden
Zusätzlich sind noch ein Statusregister, ein Status-Bereichszeiger und ein Refresh-Register vorhanden
Instruktionen müssen wortweise im Speicher abgelegt werden, bzw. ausgerichtet sein
9 Adressierungsarten
Instruktionslänge zwischen 2 und 10 Bytes
Datentypen: BIT, BCD, Byte, Worte (2 Byte), Doppelworte (32 Bit), Byteketten und Wortketten (Strings)

10.3 Der MC68000 von Motorola

Einer der »jüngsten« und leistungsfähigsten 16-Bit Mikroprozessoren, der 68000, kommt von Motorola. Entwicklungshilfen ähnlich wie bei Intel.
Der 68000 hat intern eine 32-Bit-Struktur, d.h. alle Register sind 32 Bit breit.
Zweitlieferanten sind ROCKWELL, HITACHI und THOMPSON.
Kurzdaten des 68000:
HMOS-Technologie
16 MByte Adreßraum
8 MHz Taktfrequenz
5V Spannungsversorgung
64poliges DIL-Gehäuse
Mikroprogrammiert
Es sind zwei Operationsarten möglich
- Supervisor bei der alle Instruktionen gültig sind
- User bei der bestimmte Instruktionen gesperrt sind
  (wichtig für den Aufbau von Betriebssystemen)

getrennter Adreß- und Datenbus
asynchroner Bus (handshake-Betrieb)
Wiederholung von Bus-Zyklen möglich
Selektiver Reset
acht 32-Bit Datenregister teilbar in 16-Bit- und 8-Bit-Register
sieben 32-Bit Adreßregister
1 Stackpointer der entweder als User-Stackpointer oder als Supervisor-Stackpointer fungiert
16-Bit Statusregister, davon je 8 Bit für den Benutzer (User) und das System
Instruktionen müssen wortorientiert im Speicher abgelegt sein
8 Adressierungsarten (unter anderem auto-inkrement/-dekrement)
Datentypen: BIT, BCD, Byte, Worte (16 Bit) und Doppelworte (32 Bit)

10.4 Der TMS 9900 von Texas Instruments

Der in der Literatur relativ selten genannte 16-Bit Mikroprozessor TMS9900 von Texas Instruments ist in NMOS-Technologie ausgeführt. Es ist ein etwas höherer Programmieraufwand erforderlich, da bei der Festlegung der Formate und der Adressierungsarten z.T. auf die Bitstruktur der einzelnen Befehle eingegangen werden muß.
Kurzdaten des TMS9900:
64 KByte Adreßraum
3 MHz Taktfrequenz
drei Versorgungsspannungen + 5V, + 12V und − 12V
64poliges DIL-Gehäuse
Instruktionsformat 16, 32 und 48 Bit
Datentypen: BIT, Byte und Wort (16 Bit)
8 Adressierungsarten
ein 16-Bit Statusregister mit Flags auch für z.B. logisches größer als, arithmetisches größer als, gleich u.a.
16 Register die für Daten und Adressen benutzt werden können. Durch ein »Workspace-Register« können die Register an eine beliebige Stelle im RAM gelegt werden. Es sind somit multiple Registersätze möglich
fünfzehn der Register können als Indexregister eingesetzt werden

10.5 Anwendungsschwerpunkte für 16-Bit Mikroprozessoren

Während man die 8-Bit-Typen Systemen mit geringer bis mittlerer Komplexität zuordnen kann, liegt der Einsatzschwerpunkt von 16-Bit Prozessoren in mittleren bis oberen Bereich. Besonders für Anwendungen die sehr viele Rechenoperationen erfordern, sind die »16-Bit'er« den 8-Bit Mikroprozessoren deutlich überlegen.
Beispiele für Einsatzgebiete sind Minicomputer, Steuerungen für Werkzeugmaschinen und Prozeßsteuerungen um nur einige zu nennen.

Nachwort:

Die vorgestellten Mikroprozessoren sind nur ein Teil der auf dem Markt befindlichen Typen. Jeder dieser Bausteine hat seine »Stärken« und »Schwächen« die ihn für manche Anwendungsfälle geradezu prädestinieren. Doch sollte man sich bei der Auswahl nicht nur auf die Leistungen des Bausteins selbst konzentrieren, sondern auch auf die angebotenen Entwicklungshilfen (Hard- und Softwareunterstützung). Damit überhaupt Programme in den Speicher eingegeben werden können ist schon eine Hilfe, ein sogenanntes Monitorprogramm (kurz: Monitor) erforderlich. Für kleine Programme genügt dabei schon die Hex-Eingabe. Bei größeren jedoch ist es nützlich bzw. notwendig, die Programme in mnemonischer Form einzugeben und von einem weiteren Hilfsprogramm, dem Assembler, in Maschinensprache übersetzen zu lassen.

Dies soll uns deutlich machen, daß es mit dem Prozessor allein nicht getan ist.

Für Anwendungen im Hobby-Bereich wird man sich in der Regel mit 8-Bit Mikroprozessoren begnügen, denn die 16-Bit-Bausteine übersteigen in den meisten Fällen sicher den »Hobby-Etat«. In diesem Bereich wird wichtiger sein, daß man auch preiswerte (z. T. kostenlose) Unterstützung durch Computer-Clubs für den eigenen Mikroprozessor bekommen kann. Ebenso spielen hierbei die preiswerte Verfügbarkeit von peripheren Geräten, ausbaufähige Minimalsysteme und Fachartikel eine große Rolle.

Die Auswahl ist in jedem Fall schwer. Hat man sich einmal für eine Produktfamilie entschieden, ist ein Zurück nur mit hohem finanziellen Aufwand möglich. Eine Möglichkeit mehrere Mikroprozessoren in einem System einzusetzen, wahlweise oder gleichzeitig, besteht, wenn man ein universelles Bussystem benutzt, das die eingesetzten Prozessoren gleichermaßen bedient. In den meisten Fällen lassen sich jedoch verschiedene Systeme nicht mischen, d. h. man kann z. B. vorhandene Speicherkarten nicht ohne Änderungen in zwei Systemen benutzen.

Zum Schluß noch ein positiver Aspekt.

Die »Computerei« macht Spaß und wen sie einmal gepackt hat, den läßt sie nicht mehr oder selten wieder los.

Literaturverzeichnis

1) Verschiedene Fachzeitschriften wie Elektronik, Chip, Funkschau u.a.
2) Elektronik Sonderhefte I, II, III
3) Markus Böhm, MOS-Schaltungen (TOPP 133–134)
4) Datenblätter und Applikationsschriften der Mikroprozessorhersteller

Anhang

Tabellen der meist verwendeten Potenzen zur Basis 2 und 16.

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$
$2^{11} = 2048$
$2^{12} = 4096$
$2^{13} = 8192$
$2^{14} = 16384$
$2^{15} = 32768$
$2^{16} = 65536$

$16^0 = 1$
$16^1 = 256$
$16^2 = 4096$
$16^3 = 65536$

Umrechnungstabelle von sedezimal nach dezimal und umgekehrt.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	2	3	4	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Tabelle für negative Displacements

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
E	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
D	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
C	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
B	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
A	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
9	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128

Alle Dezimalzahlen in obiger Tabelle sind negativ.

Best.-Nr. 475
ISBN 3-7724-0459-6

Dieses Buch führt den Neuling schnell und sicher in die Problematik der Mikroprozessor-Technik ein. Es vermittelt Grundlagen in übersichtlicher Form und erklärt die Funktionsweise eines Mikroprozessors anhand des einfach strukturierten Bausteins SCMP II.

Der Leser wird in die Lage versetzt, selbständig Programme für den besprochenen Prozessor zu entwickeln. Außerdem findet er beliebig erweiterbare Programmbeispiele.

Der Autor vervollständigt das Thema durch die Beschreibung gängiger Mikroprozessoren: Z80, M6809, 8085 und 6502. Natürlich werden auch die 16-Bit-Typen behandelt. Sie runden die ausgewogene Übersicht nach oben hin ab.

Dem Einsteiger ist das umfangreiche Fachwort-Lexikon aus dem Bereich der Datentechnik willkommene Arbeitshilfe und Nachschlagewerk zugleich.